Typing messages for free in security protocols: the case of equivalence properties

Rémy Chrétien, Véronique Cortier, Stéphanie Delaune

Typing messages for free in security protocols: the case of equivalence properties

Rémy Chrétien, Véronique Cortier, Stéphanie Delaune

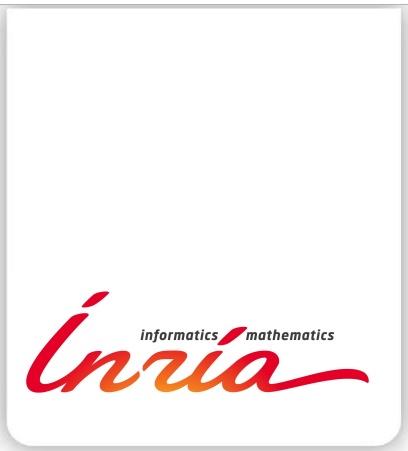
RESEARCH REPORT

N° 8546

June 16, 2014

Project-Team Cassis

ISSN 0249-6399 ISRN INRIA/RR--8546--FR+ENG



Typing messages for free in security protocols: the case of equivalence properties *

Rémy Chrétien[†‡], Véronique Cortier[‡], Stéphanie Delaune[†]

Project-Team Cassis

Research Report n° 8546 — June 16, 2014 — 43 pages

Abstract: Privacy properties such as untraceability, vote secrecy, or anonymity are typically expressed as behavioural equivalence in a process algebra that models security protocols. In this paper, we study how to decide one particular relation, namely trace equivalence, for an unbounded number of sessions.
Our first main contribution is to reduce the search space for attacks. Specifically, we show that if there is an attack then there is one that is well-typed. Our result holds for a large class of typing systems and a large class of determinate security protocols. Assuming finitely many nonces and keys, we can derive from this result that trace equivalence is decidable for an unbounded number of sessions for a class of tagged protocols, yielding one of the first decidability results for the unbounded case. As an intermediate result, we also provide a novel decision procedure in the case of a bounded number of sessions.

Key-words: formal methods, cryptographic protocols, trace equivalence

* The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement n° 258865, project ProSecure, and the ANR project JCJC VIP n^o 11 JS02 006 01.

† LSV, CNRS & ENS Cachan

‡ LORIA, CNRS & INRIA project Cassis

RESEARCH CENTRE
NANCY – GRAND EST

615 rue du Jardin Botanique
CS20101
54603 Villers-lès-Nancy Cedex

Typer les messages gratuitement dans les protocoles de sécurité: le cas des propriétés d'équivalence

Résumé : Les propriétés en lien avec le respect de la vie privée comme l'anonymat d'un vote, le secret fort, ou les propriétés de non traçabilité sont exprimées à l'aide d'équivalences observationnelles issues d'une algèbre de processus permettant de modéliser les protocoles de sécurité. Dans ce papier, nous étudions comment décider une relation d'équivalence particulière, appelée équivalence de traces, dans le cadre d'un nombre non borné de sessions.

Notre première contribution est de réduire l'espace de recherche. Plus précisément, nous montrons que si une attaque existe alors il en existe une bien typée. Notre résultat s'applique pour de nombreux systèmes de typage ainsi qu'une grande classe de protocoles déterministes. Ensuite, en supposant un nombre borné de nonces et de clefs, nous montrons que l'équivalence de traces est décidable pour un nombre non borné de sessions pour une classe de protocoles dits taggués, obtenant ainsi un des premiers résultats de décidabilité pour le cas non borné. En passant, nous fournissons aussi une nouvelle procédure de décision pour le cas d'un nombre borné de sessions.

Mots-clés : méthodes formelles, protocoles cryptographiques, équivalence de traces

1 Introduction

Privacy properties such as untraceability, vote secrecy, or anonymity are typically expressed as behavioural equivalence (*e.g.* [9, 5]). 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 [6, 21, 11]. In this paper, we study trace equivalence in the case of an unbounded number of sessions.

Our contribution. Our first main contribution is a simplification result, that reduces 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 $P \not\approx Q$ then there exists a witness which is well-typed w.r.t. P or Q, provided that P and Q are determinate processes (intuitively, messages that are outputted are completely determined by the interactions of the protocol with the environment, *i.e.* the attacker). This typing result holds for an unbounded number of sessions and an unbounded number of nonces, that is, it holds even if P and Q contain arbitrary replications and NEW operations. It holds for any typing system provided that any two unifiable encrypted subterms of P (or Q) are of the same type. It is then up to the user to adjust the typing system such that this hypothesis holds for the protocols under consideration. For simplicity, we prove this typing result for the case of symmetric encryption and concatenation but we believe that our result could be extended to the other standard cryptographic primitives.

The finer the typing system is, the more our typing result restricts the attack search. In general, our typing result does not yield directly a decidability result since even the simple property of reachability is undecidable for an unbounded number of sessions and arbitrary nonces, even if the messages are of bounded size (*e.g.* [3]). Indeed, our typing system ensures the existence of a well-typed attack (if any) but the number of well-typed traces may remain infinite. To obtain decidability, we further assume a finite number of terms of each type (*i.e.* in particular a finite number of nonces). Decidability of trace equivalence then follows from our main typing result, for a class of *simple* protocols where each subprocess uses a distinct channel (intuitively, this corresponds to session identifiers).

As an application, we consider the class of tagged protocols introduced by Blanchet and Podelski [8]. An easy way to achieve this in practice by labelling encryption and is actually a good protocol design principle [2, 18]. We show that tagged protocols induce a typing system for which trace equivalence is decidable, for simple protocols and for an unbounded number of sessions (but a fixed number of nonces).

Interestingly, the proof of our main typing result involves providing a new decision procedure for trace equivalence in the case of a bounded number of sessions. This is a key intermediate result of our proof. Trace equivalence was already shown to be decidable for a bounded number of sessions (*e.g.* [21, 11]) but we propose a novel decision procedure that further provides a *well-typed* witness whenever the two processes are not in trace equivalence. Compared to existing procedures (and in particular [21]), we

show that it is only necessary to consider unification between encrypted terms. We believe that this new decision procedure is of independent interest since it reduces the number of traces (executions) that need to be considered. Our result could therefore be used to speed up equivalence checkers like SPEC [21].

Related work. Formal methods have been very successful for the analysis of security protocols and many decision procedures and tools (*e.g.* [20, 19, 15]) have been proposed. However, most of these results focus on reachability properties such as confidentiality or authentication. Much fewer results exist for behavioural equivalences. Based on a procedure proposed by Baudet [6], a first decidability result has been proposed for determinate process without else branches, and for equational theories that capture most standard primitives [12]. Then Tiu and Dawson [21] have designed and implemented a procedure for open bisimulation, a notion of equivalence stronger than the standard notion of trace equivalence. Cheval *et al* [11] have proposed and implemented a procedure for processes with else branches and standard primitives. The tool AkisS [10] is also dedicated to trace equivalence but is not guaranteed to terminate. However, all these results focus on a bounded number of sessions. An exception is the tool ProVerif which can handle observational equivalence for an unbounded number of sessions [7]. It actually reasons on a stronger notion of equivalence (which may turn to be too strong in practice) and is again not guaranteed to terminate.

To our knowledge, the only decidability result for an unbounded number of sessions is [13]. It is shown that trace equivalence can be reduced to the equality of languages of pushdown automata. A key hypothesis for reducing to pushdown automata is that protocol rules have at most one variable, that is, at any execution step, any participant knows already every component of the message he received except for at most one component (*e.g.* a nonce received from another participant). Moreover variables shall not occur in key position, *i.e.* agents may not use received keys for encryption. This strongly limits the class of protocols that can be considered and the approach is strictly bound to this "one-variable" hypothesis. In contrast, we can consider here a much wider class of protocols, provided that they are tagged (which is easy to implement).

Our proof technique is inspired from the approach developed by Arapinis *et al* [4] for bounding the size of messages of an attack for the reachability case. Specifically, they show for some class of tagged protocols, that whenever there is an attack, there is a well-typed attack (for a particular typing system). We somehow extend their approach to trace equivalence and more general typing systems.

2 Model for security protocols

Security protocols are modelled through a process algebra inspired from [1] that manipulates terms.

2.1 Syntax

Term algebra. 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 assume a signature $\mathcal{F}$, *i.e.* a set of function symbols together with their arity. We consider:

$$\Sigma_{\mathsf{c}} = \{\mathsf{enc}, \langle\,\rangle\},\ \Sigma_{\mathsf{d}} = \{\mathsf{dec}, \mathsf{proj}_1, \mathsf{proj}_2\}, \text{ and } \Sigma = \Sigma_{\mathsf{c}} \cup \Sigma_{\mathsf{d}}.$$

The symbols dec and enc of arity 2 represent symmetric decryption/encryption. Pairing is modelled using a symbol of arity 2, denoted $\langle\ \rangle$, and projection functions are denoted proj_1 and proj_2. We further assume an infinite set of *constant symbols* Σ_0 to represent atomic data known to the attacker. The symbols in Σ_c are constructors whereas those in Σ_d are destructors. Both represent functions available to the attacker.

Given a set of A of atoms (*i.e.* names, variables, and constants), and a signature $\mathcal{F} \in \{\Sigma_\mathsf{c}, \Sigma_\mathsf{d}, \Sigma\}$, we denote by $\mathcal{T}(\mathcal{F}, \mathsf{A})$ the set of terms built from symbols in $\mathcal{F}$, and atoms in A. The subset of $\mathcal{T}(\Sigma_\mathsf{c}, \mathsf{A})$ which only contains terms with atoms as a second argument of the symbol enc, is denoted $\mathcal{T}_0(\Sigma_\mathsf{c}, \mathsf{A})$. 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 the signature Σ using (public) constants in Σ_0 as well as variables in $\mathcal{W}$, *i.e.* a term $R \in \mathcal{T}(\Sigma, \Sigma_0 \cup \mathcal{W})$. 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 relations between encryption/decryption and pairing/projections are represented through the three following rewriting rules, yielding a convergent rewrite system:

$$\mathsf{dec}(\mathsf{enc}(x, y), y) \to x, \text{ and } \mathsf{proj}_i(\langle x_1, x_2\rangle) \to x_i \text{ with } i \in \{1, 2\}.$$

Given $u \in \mathcal{T}(\Sigma, \Sigma_0 \cup \mathcal{N} \cup \mathcal{X})$, we denote by $u{\downarrow}$ its *normal form*. We refer the reader to [17] for the precise definitions of rewriting systems, convergence, and normal forms.

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

Process algebra. Let $\mathcal{C}h$ be an infinite set of *channels*. We consider processes built using the following grammar where $u \in \mathcal{T}(\Sigma_\mathsf{c}, \Sigma_0 \cup \mathcal{N} \cup \mathcal{X})$, $n \in \mathcal{N}$, and $c, c' \in \mathcal{C}h$:

$$P, Q := 0 \mid \mathsf{in}(c, u).P \mid \mathsf{out}(c, u).P \mid (P \mid Q) \mid !P \mid \mathsf{new}\, n.P \mid \mathsf{new}\, c'.\mathsf{out}(c, c').P$$

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. The process $!P$ executes P some arbitrary number of times. The name restriction "new n" is used to model the creation in a process of a fresh random number (*e.g.*, a nonce or a key) whereas channel generation "$\mathsf{new}\, c'.\mathsf{out}(c, c').P$" is used to model the creation of a new channel name that shall immediately be made public. Note that we consider only public channels. It is still useful to generate fresh (public) channel names to let the attacker identify the different sessions of a protocol (as it is often the case in practice through sessions identifiers).

We assume that names are implicitly freshly generated, thus $\mathsf{new}\, k.\mathsf{out}(c, k)$ and $\mathsf{out}(c, k)$ have exactly the same behaviour. The construction "new" becomes important in the presence of replication to distinguish whether some value k is generated at each session, *e.g.* in $!(\mathsf{new}\, k.\mathsf{out}(c, k))$ or not, *e.g.* in $\mathsf{new}\ k.(!\mathsf{out}(c, k))$.

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. For example, in the process $\mathsf{in}(c, x).\mathsf{in}(c, x)$ the variable x is bound once and thus the process is name and variable distinct. By contrast, in $\mathsf{in}(c, x) \mid \mathsf{in}(c, x)$, one occurrence

of the variable x would need to be renamed. 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.

We assume $\mathcal{C}h = \mathcal{C}h_0 \uplus \mathcal{C}h^{\mathsf{fresh}}$ where $\mathcal{C}h_0$ and $\mathcal{C}h^{\mathsf{fresh}}$ are two infinite and disjoint sets of channels. Intuitively, channels of $\mathcal{C}h^{\mathsf{fresh}}$, denoted $ch_1, \ldots, ch_i, \ldots$ will be used in the semantics to *instantiate* the channels generated during the execution of a protocol. They shall not be part of its specification.

Definition 1 *A* protocol P *is a process such that* P *is ground,* i.e. $fv(P) = \emptyset$*;* P *is name and variable distinct; and* P *does not use channel names from* $\mathcal{C}h^{\mathsf{fresh}}$.

Example 2 *The Otway-Rees protocol [14] 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. We use restricted channels to model the use of unique session identifiers used along an execution of the protocol. 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* $\langle x_1, \langle \ldots \langle x_{n-1}, x_n\rangle\rangle\rangle$.

$$P_{\mathsf{OR}} = !\,\mathsf{new}\ c_1.\mathsf{out}(c_A, c_1).P_A \mid !\,\mathsf{new}\ c_2.\mathsf{out}(c_B, c_2).P_B \mid !\,\mathsf{new}\ c_3.\mathsf{out}(c_S, c_3).P_S$$

where the processes P_A*,* P_B *are given below, and* P_S *can be defined in a similar way.*

$$P_A = \mathsf{new}\ m.\mathsf{new}\ n_a.\ \mathsf{out}(c_1, \langle m, \mathsf{a}, \mathsf{b}, \mathsf{enc}(\langle n_a, m, \mathsf{a}, \mathsf{b}\rangle, k_{as})\rangle).\ \mathsf{in}(c_1, \langle m, \mathsf{enc}(\langle n_a, x_{ab}\rangle, k_{as})\rangle);$$

$$P_B = \mathsf{in}(c_2, \langle y_m, \mathsf{a}, \mathsf{b}, y_{as}\rangle).\mathsf{new}\ n_b.\mathsf{out}(c_2, \langle y_m, \mathsf{a}, \mathsf{b}, y_{as}, \mathsf{enc}(\langle n_b, y_m, \mathsf{a}, \mathsf{b}\rangle, k_{bs})\rangle).\ \mathsf{in}(c_2, \langle y_m, z_{as}, \mathsf{enc}(\langle n_b, y_{ab}\rangle, k_{bs})\rangle).\mathsf{out}(c_2, \langle y_m, z_{as}\rangle)$$

2.2 Semantics

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

- $\mathcal{P}$ 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, *i.e.* terms in $\mathcal{T}_0(\Sigma_{\mathsf{c}}, \Sigma_0 \cup \mathcal{N})$.

We often write P instead of $(\{P\}; \emptyset)$, and $P \cup \mathcal{P}$ or $P \mid \mathcal{P}$ instead of $\{P\} \cup \mathcal{P}$. The terms in ϕ represent the messages that are known by the attacker. The operational semantics of a process is induced by the relation $\xrightarrow{\alpha}$ over configurations defined below.

$(\mathsf{in}(c,u).P \cup \mathcal{P};\phi) \xrightarrow{\mathsf{in}(c,R)} (P\sigma \cup \mathcal{P};\phi)$ where R is a recipe such that $R\phi\!\downarrow$ is a message and $R\phi\!\downarrow = u\sigma$ for some σ with $dom(\sigma) = vars(u)$

$(\mathsf{out}(c,u).P \cup \mathcal{P};\phi) \xrightarrow{\mathsf{out}(c,\mathsf{w}_{i+1})} (P \cup \mathcal{P};\phi \cup \{\mathsf{w}_{i+1} \triangleright u\})$ where u is a message and i is the number of elements in ϕ

$(\mathsf{new}\, c'.\mathsf{out}(c,c').P \cup \mathcal{P};\phi) \xrightarrow{\mathsf{out}(c,ch_i)} (P\{^{ch_i}/_{c'}\} \cup \mathcal{P};\phi)$ where ch_i is the "next" fresh channel name available in $\mathcal{C}h^{\mathsf{fresh}}$

$(\mathsf{new}\ n.P \cup \mathcal{P};\phi) \xrightarrow{\tau} (P\{^{n'}/_{n}\} \cup \mathcal{P};\phi)$ where n' is a fresh name in $\mathcal{N}$

$(!P \cup \mathcal{P};\phi) \xrightarrow{\tau} (P \cup !P \cup \mathcal{P};\phi)$

The first rule allows the attacker to send to some process a term built from publicly available terms and symbols. The second rule 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 an encryption with a non atomic key, the evaluation fails and there is no output. The third rule corresponds to the special case of an output of a freshly generated channel name. In such a case, the channel is not added to the frame but it is implicitly assumed known to the attacker, as all the channel names. These three rules are the only observable actions. 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 $K \stackrel{\mathsf{tr}}{\Longrightarrow} K'$ 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. For every protocol P, we define its *set of traces* as follows:

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

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

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

$$\begin{aligned}\mathsf{tr} = \quad & \mathsf{out}(c_A, ch_1).\mathsf{out}(c_B, ch_2).\mathsf{out}(ch_1, \mathsf{w}_1).\mathsf{in}(ch_2, \mathsf{w}_1).\\ & \mathsf{out}(ch_2, \mathsf{w}_2).\mathsf{in}(ch_2, R_0).\mathsf{out}(ch_2, \mathsf{w}_3).\mathsf{in}(ch_1, \mathsf{w}_3)\end{aligned}$$

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. Actually such a sequence of actions allows one to reach the following frame with* $t_{\mathsf{enc}} = \mathsf{enc}(\langle n_a, m, \mathsf{a}, \mathsf{b}\rangle, k_{as})$*:*

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

We have that $(\mathsf{tr},\phi) \in \mathsf{trace}(P_{\mathsf{OR}})$*. The first five actions actually correspond to a normal execution of the protocol. Then, the agent who plays* P_B *will accept in input the message built using* R_0*,* i.e. $u = \langle m, \mathsf{enc}(\langle n_a, m, \mathsf{a}, \mathsf{b}\rangle, k_{as}), \mathsf{enc}(\langle n_b, m, \mathsf{a}, \mathsf{b}\rangle, k_{bs})\rangle$*. Indeed, this message has the expected form. At this stage, the agent who plays* P_B *is waiting for a message of the form:* $u_0 = \langle m, z_{as}, \mathsf{enc}(\langle n_b, y_{ab}\rangle, k_{bs})\rangle$*. The substitution* $\sigma = \{z_{as} \triangleright t_{\mathsf{enc}},\ y_{ab} \triangleright \langle m, \mathsf{a}, \mathsf{b}\rangle\}$ *is such that* $u = u_0\sigma$*. Once this input has been done, a message is outputted (action* $\mathsf{out}(ch_3, \mathsf{w}_3)$*) and given in input to* P_A *(action* $\mathsf{in}(ch_1, \mathsf{w}_3)$*).*

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* $\langle m, \mathsf{a}, \mathsf{b}\rangle$*.*

2.3 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 [9]. We first introduce a notion of intruder's knowledge well-suited to cryptographic primitives for which the success of decrypting is visible.

Definition 2 *Two frames ϕ_1 and ϕ_2 are* statically equivalent*, $\phi_1 \sim \phi_2$, when we have that $dom(\phi_1) = dom(\phi_2)$, and:*

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

Intuitively, two frames are 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 $R\phi_1\downarrow = m \in \mathcal{T}_0(\Sigma_c, \Sigma_0 \cup \mathcal{N})$, but $R\phi_2\downarrow = \mathsf{proj}_1(n) \notin \mathcal{T}_0(\Sigma_c, \Sigma_0 \cup \mathcal{N})$, hence $\phi_1 \not\sim \phi_2$. This non static equivalence can also be established 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 $R_1\phi_2\downarrow \neq R_2\phi_2\downarrow$.*

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

Definition 3 *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 $\phi \sim \phi'$. The protocols P and Q are* trace equivalent*, written $P \approx Q$, if $P \sqsubseteq Q$ and $Q \sqsubseteq P$.*

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 $P = \mathsf{in}(c, x).\mathsf{out}(c, \mathsf{enc}(n, k))$ as well as the protocol $Q = \mathsf{in}(c, x).\mathsf{out}(c, \mathsf{enc}(\mathsf{enc}(n, x), k))$. 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.*

In what follows, we consider *determinate* protocols as defined in [10], *i.e.*, we consider protocols in which the attacker knowledge is completely determined (up to static equivalence) by its past interaction with the protocol participants.

Definition 4 *A protocol P is* determinate *if for any tr, and for any $(\mathcal{P}_1, \phi_1)$, $(\mathcal{P}_2, \phi_2)$ such that $P \xRightarrow{\mathsf{tr}} (\mathcal{P}_1, \phi_1)$, and $P \xRightarrow{\mathsf{tr}} (\mathcal{P}_2, \phi_2)$, we have that $\phi_1 \sim \phi_2$.*

Assume given two determinate protocols P and Q such that $P \not\sqsubseteq Q$. A *witness of non-inclusion* is a trace tr for which there exists ϕ such that $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ and:

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

A *witness of non-equivalence* for determinate protocols P and Q is a trace tr that is a witness for $P \not\sqsubseteq Q$ or $Q \not\sqsubseteq P$. Note that when a protocol P is determinate, once the sequence tr is fixed, all the frames reachable through tr are actually in static equivalence, which ensures the unicity of ϕ', if it exists, up-to static equivalence.

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 $\mathsf{out}(c_1, x_{ab})$ at the end of the process P_A;*
- *P^2_{OR} is as P_{OR} but we add the instruction $\mathsf{new}\, n.\mathsf{out}(c_1, n)$ at the end of 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 $\mathsf{tr}' = \mathsf{tr}.\mathsf{out}(ch_1, \mathsf{w}_4)$ 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, $\phi_1 \not\sim \phi_2$ 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 [14].

3 Existence of a well-typed witness of non-equivalence

In this section, we present our first main contribution: a simplification result that reduces the search space for attacks. Roughly, when looking for an attack, we can restrict ourselves to consider well-typed traces. This results holds for a general class of typing systems and as soon as the protocols under study are determinate and type-compliant. We first explain these hypotheses and then we state our general simplification result (see Theorem 1). The proof of this simplification result involves to provide a novel decision procedure for trace equivalence in the case of a bounded number of sessions. The novelty of this decision procedure, in comparison to the existing ones, is to provide a well-typed witness whenever the two processes are not in trace equivalence. This key intermediate result is stated in Proposition 1.

3.1 Typing system

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 5 *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 $t \in \mathcal{T}(\Sigma_{\mathsf{c}}, \Sigma_0 \cup \mathcal{N} \cup \mathcal{X})$ 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. We present in Section 5 a typing system that allows us to reduce the search space (and then derive decidability) for a large subclass of (tagged) protocols.

3.2 Well-typed trace

Whether or not a trace is well-typed is defined w.r.t. the set of *symbolic traces* of a protocol. Formally, we define $\xrightarrow{\mathsf{tr}_s}_s$ to be the transitive closure of the relation $\xrightarrow{\alpha_s}_s$ defined between processes as follows:

$$\mathsf{in}(c,u).P \cup \mathcal{P} \xrightarrow{\mathsf{in}(c,u)}_s P \cup \mathcal{P} \qquad !P \cup \mathcal{P} \xrightarrow{\tau}_s P' \cup !P \cup \mathcal{P}$$

$$\mathsf{out}(c,u).P \cup \mathcal{P} \xrightarrow{\mathsf{out}(c,u)}_s P \cup \mathcal{P} \qquad \mathsf{new}\, n.P \cup \mathcal{P} \xrightarrow{\tau}_s P\{^{n'}/_{n}\} \cup \mathcal{P}$$

$$\mathsf{new}\, c'.\mathsf{out}(c,c').P \cup \mathcal{P} \xrightarrow{\mathsf{out}(c,ch_i)}_s P\{^{ch_i}/_{c'}\} \cup \mathcal{P}$$

where P' is equal to P up to renaming of variables that do not occur yet in the trace with fresh ones (of the same type), n' is a fresh name (of the same type as n), and ch_i is the "next" fresh channel name available in $\mathcal{C}h^{\mathsf{fresh}}$.

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

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

Intuitively, the symbolic traces are simply all possible traces before instantiation of the variables, with some renaming to avoid unwanted captures.

Example 7 *Let $P_1 = \mathsf{in}(c,x).!\mathsf{new}\, k.\, \mathsf{in}(c, \mathsf{enc}(\langle x, y\rangle, k))$. We have that:*

$$\mathsf{tr}_s = \mathsf{in}(c,x).\mathsf{in}(c, \mathsf{enc}(\langle x, y_1\rangle, k_1)).\mathsf{in}(c, \mathsf{enc}(\langle x, y_2\rangle), k_2) \in \mathsf{trace}_s(P_1)$$

Indeed, the variable x is bound before replication.

As stated in the lemma below, any concrete trace is the instance of a symbolic trace.

Lemma 1 *Let P be a protocol and $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$. We have that $\mathsf{tr}\phi{\downarrow} = \mathsf{tr}_s\sigma$ for some $\mathsf{tr}_s \in \mathsf{trace}_s(P)$ and some substitution σ.*

A well-typed trace is simply a trace that is well-typed w.r.t. one of the symbolic traces. Since keys are atomic, some executions may fail when a protocol is about to output a term that contains an encryption with a non atomic key. To detect these behaviours, we need to consider slightly ill-typed traces. Formally, we consider a special constant $\omega \in \Sigma_0$. Its usefulness is illustrated in Example 8.

Definition 6 *A* first-order trace *of P is a sequence $\mathsf{tr} = \mathsf{tr}_s\sigma$ where $\mathsf{tr}_s \in \mathsf{trace}_s(P)$ and σ is a substitution such that for any $\mathsf{io}(c,u)$ that occurs in tr_s with $\mathsf{io} \in \{\mathsf{in}, \mathsf{out}\}$ and u not a channel, then $u\sigma \in \mathcal{T}_0(\Sigma_c, \Sigma_0 \cup \mathcal{N} \cup \mathcal{X})$. The trace tr is said to be:*

- well-typed *w.r.t. a typing system* $(\mathcal{T}, \delta)$ *if there exists such a* σ *that is well-typed;*
- pseudo-well-typed *w.r.t. a typing system* $(\mathcal{T}, \delta)$ *if there exists such* σ*, as well as* $\mathsf{c}_0 \in \Sigma_0$ *and* σ' *such that* $\sigma = \sigma'\{^{\langle\omega,\omega\rangle}/_{\mathsf{c}_0}\}$ *with* σ' *well-typed.*

Then a trace $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ *is* well-typed *(resp.* pseudo-well-typed*) if* $\mathsf{tr}\phi\!\downarrow$ *is well-typed (resp.* pseudo-well-typed*).*

Note that Lemma 1 ensures that $\mathsf{tr}\phi\!\downarrow$ is a first-order trace of P, and a well-typed trace is also pseudo-well-typed.

Example 8 *Going back to Example 5, let* $\mathsf{tr} = \mathsf{in}(c, \langle\omega, \omega\rangle).\mathsf{out}(c, \mathsf{w}_1)$*. We have that* $(\mathsf{tr}, \{\mathsf{w}_1 \triangleright \mathsf{enc}(n, k)\}) \in \mathsf{trace}(P)$ *while there exists no frame* ψ *such that* $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$*. Consider the typing system* $(\mathcal{T}, \delta)$ *such that* $\delta(t) = \mathsf{atom}$ *for any atom or variable* t *and* $\delta(t) = \neg\mathsf{atom}$ *if* t *is not an atom. We can see there exists no well-typed witness of* $P \not\approx Q$ *(while* P *and* Q *are type-compliant as defined in Definition 7). However, the witness* $(\mathsf{tr}, \{\mathsf{w}_1 \triangleright \mathsf{enc}(n, k)\})$ *of* $P \not\sqsubseteq Q$ *is pseudo-well-typed (note that* $\langle\omega, \omega\rangle$ *occurs in* tr*). Intuitively, pseudo-well-typed traces harness the ability for the attacker to use the protocol as an oracle to test if some terms (when used in a key position) are atomic.*

3.3 Type compliance

Our main assumption on the typing of protocols is that any two unifiable encrypted subterms are of the same type. The goal of this part is to state this hypothesis formally.

Due to the presence of replication, we need to consider two copies of protocols in order to consider different instances of the variables. Given a protocol P with replication, we define its 2-unfolding $\mathsf{unfold}^2(P)$ to be the protocol such that every occurrence of a process $!R$ in P is replaced by $R \mid R$, and some α-renaming is performed on one copy to ensure names and variables distinctness of the resulting process. Note that if P is a protocol that does not contain any replication, we have that $\mathsf{unfold}^2(P) = P$.

Example 9 *Let* P_1 *be the protocol defined in Example 7. We have that:*
$\mathsf{unfold}^2(P_1) = \mathsf{in}(c, x).(\mathsf{new}\, k_1.\mathsf{in}(c, \mathsf{enc}(\langle x, y_1\rangle, k_1)) \mid \mathsf{new}\, k_2.\mathsf{in}(c, \mathsf{enc}(\langle x, y_2\rangle, k_2)))$

We write $St(t)$ for the set of *(syntactic) subterms* of a term t, and $ESt(t)$ the set of its *encrypted subterms*, *i.e.* $ESt(t) = \{u \in St(t) \mid u \text{ is of the form } \mathsf{enc}(u_1, u_2)\}$. 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* $t, t' \in ESt(\mathsf{unfold}^2(P))$ *we have that:* t *and* t' *unifiable implies that* $\delta(t) = \delta(t')$*.*

3.4 Main result

We are now ready to state our first main contribution: if there is an attack, then there is a pseudo-well-typed attack. This result holds for protocols with replications and nonces.

Theorem 1 *Let* P *and* Q *be two determinate protocols type-compliant w.r.t.* $(\mathcal{T}_1, \delta_1)$ *and* $(\mathcal{T}_2, \delta_2)$ *respectively. We have that* $P \not\approx Q$ *if, and only if, there exists a witness of non-equivalence* tr *such that:*

- *either* $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ *for some* ϕ *and* (tr, ϕ) *is pseudo-well-typed w.r.t.* $(\mathcal{T}_1, \delta_1)$*;*

- *or* $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$ *for some* ψ *and* (tr, ψ) *is pseudo-well-typed w.r.t.* $(\mathcal{T}_2, \delta_2)$.

The key step for proving Theorem 1 is to provide a decision procedure, in the bounded case (*i.e.* processes without replication), that returns a pseudo-well-typed witness of non-equivalence.

Proposition 1 *Let P and Q be two determinate protocols without replication. There exists an algorithm that decides whether $P \approx Q$ and if not, returns a witness* tr *of non-equivalence. Moreover, if P and Q are type-compliant w.r.t. $(\mathcal{T}_1, \delta_1)$ and $(\mathcal{T}_2, \delta_2)$ respectively, the witness* tr *of non-equivalence returned by the algorithm is such that:*

- *either* $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ *for some* ϕ *and* (tr, ϕ) *is pseudo-well-typed w.r.t.* $(\mathcal{T}_1, \delta_1)$*;*

- *or* $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$ *for some* ψ *and* (tr, ψ) *is pseudo-well-typed w.r.t.* $(\mathcal{T}_2, \delta_2)$.

This algorithm is presented in Appendix A, and its properties are proven in Appendices B and C. The main idea is to assume given a decision procedure (for a bounded number of sessions) for reachability properties such as those proposed in [19, 15, 22] and to built on top of it a decision procedure for trace equivalence. Our procedure is carefully design to only allow unification between encrypted subterms. To achieve this,

1. we use as a reachability blackbox one that satisfies this requirement. Most of the existing algorithms (*e.g.* [19, 15, 22]) were not designed with such a goal in mind. However, in the case of the algorithm given in [15], it has already been shown how it can be turned into one that satisfies this requirement [16].

2. we design carefully the remaining of our algorithm to only consider unification between encrypted subterms.

This design allows us to provide a pseudo-well-typed witness when the protocols under study are type-compliant and not trace equivalent.

Then, relying on Proposition 1, the proof of Theorem 1 is almost immediate. Indeed, whenever two determinate type-compliant protocols P and Q are not in trace equivalence, there exists a witness of non-inclusion for $P \sqsubseteq Q$ (or $Q \sqsubseteq P$) for a bounded version of P and Q (unfolding the replications).

4 Decidability result

Now, assuming finitely many terms of each type, and in particular finitely many nonces, we obtain a new decidability result for trace equivalence, for an unbounded number of sessions. Compared to [13], we no longer need to restrict the number of variables per transition (to one), we allow variables in key positions, and we are more flexible in the control-flow of the program (we may have arbitrary sequences of in and out actions).

4.1 Simple processes

To establish decidability, we consider the class of simple protocols as given in [12] but we do not allow name restriction. Intuitively, simple protocols are protocols such that each copy of a replicated process has its own channel. This reflects the fact that due to IP addresses and sessions identifiers, an attacker can identify which process and which session he is sending messages to (or receiving messages from).

Definition 8 *A* simple protocol P *is a protocol of the form* $P_U \mid P_B$ *where:*

- $P_U = !\mathsf{new}\ c'_1.\mathsf{out}(c_1, c'_1).B_1 \mid \ldots \mid !\mathsf{new}\ c'_m.\mathsf{out}(c_m, c'_m).B_m$*; and*
- $P_B = B_{m+1} \mid \ldots \mid B_{m+n}$.

Each B_i *with* $1 \leq i \leq m$ *(resp.* $m < i \leq m+n$*) is a ground process on channel* c'_i *(resp.* c_i*) built using the following grammar:*

$$B := 0 \mid \mathsf{in}(c'_i, u).B \mid \mathsf{out}(c'_i, u).B \text{ where } u \in \mathcal{T}_0(\Sigma_\mathsf{c}, \Sigma_0 \cup \mathcal{N} \cup \mathcal{X}).$$

Moreover, we assume that $c_1, \ldots, c_n, c_{n+1}, \ldots, c_{n+m}$ *are pairwise distinct.*

Example 10 *The protocol presented in Example 2 is not simple yet: we need to consider only finitely many nonces. To achieve this, we may remove all the instructions* "$\mathsf{new}\ n$" *with* $n \in \mathcal{N}$ *that occur in the process. Note that removing for instance* "$\mathsf{new}\ n_a$" *from the process* P_A *means that* n_a *is still modelled as a name, and thus it is unknown to the attacker. However, we do not assume anymore that a fresh nonce is generated at each session.*

Simple protocols form a large class of protocols that are determinate: the attacker knows exactly who is sending a message or from whom he is receiving a message. Actually, given a simple protocol P and a sequence of observable actions tr, there is a unique configuration $(\mathcal{P}; \phi)$ (up to some internal reduction steps) such that $P \stackrel{\mathsf{tr}}{\Longrightarrow} (\mathcal{P}; \phi)$.

Lemma 2 *A simple protocol is determinate.*

4.2 Main result

Our decidability result relies on the assumption that there are finitely many terms of each type (of the protocol), once the number of constants is bound for each type.

Formally, we say that a typing system $(\mathcal{T}, \delta)$ is *finite* if, for any set $\mathsf{A} \subseteq \mathcal{N} \cup \Sigma_0$ such that there is a finite number of names/constants of each type, then there are finitely many terms of each type, that is, for any $\tau \in \mathcal{T}$, the following set is finite and computable:

$$\{t \in \mathcal{T}(\Sigma_\mathsf{c}, \mathsf{A}) \mid \delta(t) = \tau\}.$$

Theorem 2 *The problem of deciding whether two simple protocols* P *and* Q*, type-compliant w.r.t. some finite typing systems* $(\mathcal{T}_1, \delta_1)$ *and* $(\mathcal{T}_2, \delta_2)$ *are trace equivalent (*i.e. $P \approx Q$*) is decidable.*

Proof. (Sketch) Since simple protocols are determinate (see Lemma 2), we obtain, thanks to our typing result (Theorem 1), the existence of well-typed witness of non-equivalence when such a witness exists. We further show that we can bound the number of useful constants in the witness trace. We then derive from the finiteness of the typing system that the witness trace uses finitely many distinct terms. Therefore, after some point, the trace only reproduces already existing transitions. Using the form of simple protocols, we can then show how to shorten the length of the witness trace. □

5 Application: tagged protocols

In this section, we instantiate our general results (Theorems 1 and 2) by exhibiting a class of protocols that is type-compliant for rather fine-grained typing systems. We consider tagged protocols, for a notion of tagging similar to one introduced by Blanchet [8].

Assume given a protocol P and an unfolding P' of it (remember that when computing $\mathsf{unfold}^2(P)$ names and variables are renamed to avoid clashes). Let u be a term in $\mathcal{T}(\Sigma_c, \Sigma_P \cup \mathcal{N}'_P \cup \mathcal{X}'_P)$ where $\Sigma_P, \mathcal{N}'_P, \mathcal{X}'_P$ are the constants, names, and variables occurring in P', we denote by $\overline{u}$ the transformation that replaces any name and variable occurring in u by its representative in $\mathcal{N}_P$ and $\mathcal{X}_P$ where $\mathcal{N}_P$ and $\mathcal{X}_P$ are the names and variables occurring in P.

Definition 9 *A protocol P is* tagged *if there exists a substitution σ_P such that for any $s_1, s_2 \in ESt(\mathsf{unfold}^2(P))$ with s_1 and s_2 unifiable, we have that $\overline{s_1\sigma_P} = \overline{s_2\sigma_P}$.*

Tagging can easily be enforced by labelling encrypted terms, as proposed in [8].

Definition 10 *A protocol P is* strongly tagged *if:*

1. *any term in $ESt(P)$ is of the form $\mathsf{enc}(\langle \mathsf{c}, m\rangle, k)$ for some $\mathsf{c} \in \Sigma_0$; and*
2. *there exists σ_P such that for any $s, t \in ESt(P)$ with $s = \mathsf{enc}(\langle \mathsf{c}_0, s_1\rangle, s_2)$ and $t = \mathsf{enc}(\langle \mathsf{c}_0, t_1\rangle, t_2)$ for some $\mathsf{c}_0 \in \Sigma_0$, we have that $s\sigma_P = t\sigma_P$.*

The second condition requires that there is a a substitution that unifies any two tagged terms unless their tags differ. This condition is easy to achieve for executable protocols. More precisely, assume a protocol admits an execution where each protocol step (in and out) is executed once (*i.e.* there is one honest execution). This protocol can be easily strongly tagged by adding a distinct tag in each encrypted term.

Lemma 3 *Let P be a protocol. If P is strongly tagged then P is tagged.*

Example 11 *In our modelling of the Otway-Rees protocol, the protocols P^1_{OR} and P^2_{OR} (as described in Example 6) are not tagged. For instance, consider the terms $s_1 = \mathsf{enc}(\langle n_a, m, \mathsf{a}, \mathsf{b}\rangle, k_{as})$ and $s_2 = \mathsf{enc}(\langle n_a, x_{ab}\rangle, k_{as})$. Both are encrypted subterms of P_A (and thus of $\mathsf{unfold}^2(P^1_{\mathsf{OR}})$ and $\mathsf{unfold}^2(P^2_{\mathsf{OR}})$) and s_1 and s_2 are unifiable. Now, let $s_3 = \mathsf{enc}(\langle z_a, k_{ab}\rangle, k_{as})$. Actually, s_3 is an encrypted subterm of P_S which is unifiable with s_2. However, there exists no substitution σ such that $\overline{s_1\sigma} = \overline{s_2\sigma} = \overline{s_3\sigma}$.*

We can consider a tagged, and safer, version of the Otway-Rees protocol by introducing 4 different tags, denoted 1,2,3 and 4, that are modelled using constants from Σ_0.

$$P'_{\mathsf{OR}} = !\,\mathsf{new}\ c_1.\mathsf{out}(c_A, c_1).P'_A \mid !\,\mathsf{new}\ c_2.\mathsf{out}(c_B, c_2).P'_B \mid !\,\mathsf{new}\ c_3.\mathsf{out}(c_S, c_3).P'_S$$

$$\begin{aligned}
P'_A = \ & \mathsf{new}\ m.\mathsf{new}\ n_a.\ \mathsf{out}(c_1, \langle m, \mathsf{a}, \mathsf{b}, \mathsf{enc}(\langle 1, n_a, m, \mathsf{a}, \mathsf{b}\rangle, k_{as})\rangle).\\
& \mathsf{in}(c_1, \langle m, \mathsf{enc}(\langle 2, n_a, x_{ab}\rangle, k_{as})\rangle)\\
P'_B = \ & \mathsf{in}(c_2, \langle y_m, \mathsf{a}, \mathsf{b}, y_{as}\rangle).\\
& \mathsf{new}\ n_b.\ \mathsf{out}(c_2, \langle y_m, \mathsf{a}, \mathsf{b}, y_{as}, \mathsf{enc}(\langle 3, n_b, y_m, \mathsf{a}, \mathsf{b}\rangle, k_{bs})\rangle).\\
& \mathsf{in}(c_2, \langle y_m, z_{as}, \mathsf{enc}(\langle 4, n_b, y_{ab}\rangle, k_{bs})\rangle).\mathsf{out}(c_2, \langle y_m, z_{as}\rangle)\\
P'_S = \ & \mathsf{in}(c_3, \langle z_m, \mathsf{a}, \mathsf{b}, \mathsf{enc}(\langle 1, z_a, z_m, \mathsf{a}, \mathsf{b}\rangle, k_{as}), \mathsf{enc}(\langle 3, z_b, z_m, \mathsf{a}, \mathsf{b}\rangle, k_{bs})\rangle).\\
& \mathsf{new}\ k_{ab}.\ \mathsf{out}(c_3, \langle z_m, \mathsf{enc}(\langle 2, z_a, k_{ab}\rangle, k_{as}), \mathsf{enc}(\langle 4, z_b, k_{ab}\rangle, k_{bs})\rangle)
\end{aligned}$$

and P'^1_{OR} and P'^2_{OR} are defined similarly as P^1_{OR} and P^2_{OR} relying on P'_{OR} instead of P_{OR}. Note that tr' is no longer a witness of $P'^1_{\mathsf{OR}} \not\approx P'^2_{\mathsf{OR}}$ as the attack has been removed

by this tagging scheme. We can show that P'_{OR} is strongly tagged: consider the natural execution of P'_{OR}, matching inputs and outputs as intended. From this execution we can define:

$$\sigma_P = \{x_{ab} \triangleright k_{ab},\ y_m \triangleright m,\ y_{as} \triangleright \mathsf{enc}(\langle 1, n_a, m, \mathsf{a}, \mathsf{b}\rangle, k_{as})\rangle,\\ z_{as} \triangleright \mathsf{enc}(\langle 2, n_a, k_{ab}\rangle, k_{as}),\ z_m \triangleright m,\ z_a \triangleright n_a,\ z_b \triangleright n_b\}.$$

It is then easy to check that for any two terms s_1 and s_2 that are unifiable, their instances by σ_P are actually identical.

For any tagged protocol, we can infer a finite typing system, and show the type-compliance of the tagged protocol w.r.t. this typing system. Thus, relying on Theorem 2, we derive the following decidability result for simple and tagged protocols.

Corollary 1 *The problem of deciding whether two simple and tagged protocols P and Q are trace equivalent (*i.e. *$P \approx Q$) is decidable.*

Proof. (Sketch) The first step of the proof consists in associating to a tagged protocol P, a typing system $(\mathcal{T}_P, \delta_P)$ such that P is type-compliant w.r.t. $(\mathcal{T}_P, \delta_P)$. Intuitively, $(\mathcal{T}_P, \delta_P)$ is simply induced by σ_P, the substitution ensuring the tagged condition in Definition 9. For example, the type of a closed term t is t itself while the type of a variable x in P is simply $x\sigma_P$. This definition is then propagated to any term. With such typing systems, we can show that the size of a term (*i.e.* number of function symbols) is smaller than the size "indicated" by its type (*i.e.* the size of the type, viewed as a term). Thus the typing system $(\mathcal{T}_P, \delta_P)$ is finite. We then conclude by applying Theorem 2. □

Example 12 *Consider the protocols $\overline{P'^1_{\mathsf{OR}}}$ and $\overline{P'^2_{\mathsf{OR}}}$ obtained from P'^1_{OR} and P'^2_{OR} by removing the instructions corresponding to a name restriction. These protocols are still strongly tagged and are now simple. Thus, our algorithm can be used to check whether these two protocols are in trace equivalence or not. This equivalence actually models a notion of strong secrecy of the key received by A. Since we have bounded the number of nonces, this equivalence does not require that the key is renewed at each session but it requires the key to be indistinguishable from a (private) name, n in our setting.*

6 Conclusion

Decidability results for unbounded nonces are rare and complex, even in the reachability case. One of the only results has been established by Ramanujam and Suresh [20], assuming a particular tagging scheme (which itself involves nonces). We plan to explore whether our typing result could be applied to the tagging scheme defined in [20], to derive decidability of trace equivalence in the presence of nonces.

Our main typing result relies on the design of a new procedure in the case of a bounded number of sessions, that preserves typing. Specifically, we show that it is sufficient to consider only unification between encrypted (sub)terms. We think that this result can be applied to existing decision procedures (in particular SPEC [21] and also APTE [11], with some more work) to speed up their corresponding tools. As future work, we plan to implement this optimisation and measure its benefit.

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. Abadi and R. M. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Trans. Software Eng.*, 22(1):6–15, 1996.

[3] R. Amadio and W. Charatonik. On name generation and set-based analysis in the Dolev-Yao model. In *13th Int. Conference on Concurrency Theory (CONCUR'02)*, 2002.

[4] M. Arapinis and M. Duflot. Bounding messages for free in security protocols. In *27th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'07)*, 2007.

[5] 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.

[6] 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.

[7] B. Blanchet, M. Abadi, and C. Fournet. Automated Verification of Selected Equivalences for Security Protocols. In *20th Symposium on Logic in Computer Science*, 2005.

[8] B. Blanchet and A. Podelski. Verification of cryptographic protocols: Tagging enforces termination. In *Foundations of Software Science and Computation Structures (FoSSaCS'03)*.

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

[10] R. Chadha, Ş. Ciobâcă, and S. Kremer. Automated verification of equivalence properties of cryptographic protocols. In *21th European Symposium on Programming (ESOP'12)*, LNCS.

[11] 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.

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

[13] R. Chrétien, V. Cortier, and S. Delaune. From security protocols to pushdown automata. In *40th Int. Colloquium on Automata, Languages and Programming (ICALP'13)*, 2013.

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

[15] 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.

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

[17] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*. Elsevier, 1990.

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

[19] 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.

[20] R. Ramanujam and S. P. Suresh. Tagging makes secrecy decidable with unbounded nonces as well. In *23rd Conference of Foundations of Software Technology and Theoretical Computer Science (FSTTCS'03)*, LNCS, pages 363–374. Springer, 2003.

[21] 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.

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

Section A aims at introducing the decision procedure needed to prove Proposition 1 and Theorem 1. Section B will focus on the proof of completeness of the said procedure, as stated in Proposition 4 later in the appendix. Section C will provide the complete proofs of Proposition 1 and Theorem 1. Section D will prove Theorem 2 and Section E will finally prove Corollary 1.

In the following, the application of a substitution σ to a term u will be written $u\sigma$. We denote $dom(\sigma)$ its *domain*, and $img(\sigma)$ its *image*. A *ground* substitution is a substitution such that $vars(u) = \emptyset$ for each $u \in img(\sigma)$.

A A type preserving decision algorithm for bounded processes

In this section, we provide a new decision procedure for trace equivalence in the case of a bounded number of sessions to prove Proposition 1, and is thus pivotal in proving Theorem 1. The novelty of this procedure is to provide a well-typed witness whenever the two protocols are not in trace equivalence. A bounded number of sessions means formally that we consider the case of *bounded protocols*, namely determinate protocols without replication and thus without name restriction.

A.1 Reachability blackbox

The main idea is to assume given a decision procedure (for a bounded number of sessions) for reachability properties. Several decision procedures have already been proposed [19, 15, 22]. They are based on constraint solving techniques and even if they differ on the way the constraints are solved, the basic ideas are actually the same. These decision procedures actually do not simply say whether some state is reachable or not. They also provide a finite representation of all possible executions. More precisely, these algorithms compute a finite set of first-order (symbolic) traces that are in *solved form*, *i.e.* such that these traces are actually valid first-order traces when the variables are interpreted as constants.

Definition 11 *Let* $\mathsf{tr}_s = \mathsf{io}_1(c_1, u_1) \dots \mathsf{io}_n(c_n, u_n)$ *be a first-order trace of* P. *Its* associated frame *is*

$$\phi_s = \{\mathsf{w}_1 \triangleright u_{i_1}, \dots, \mathsf{w}_\ell \triangleright u_{i_\ell}\}.$$

where $i_1 \dots i_\ell$ *is the increasing sequence of indices that captures all the outputs of terms of the trace* tr_s, i.e. *such that* $\{i_1, \dots, i_\ell\} = \{j \mid \mathsf{io}_j = \mathsf{out}$ *and* u_j *is not a channel*$\}$

Definition 12 *A first-order trace* $\mathsf{tr}_s = io_1, \dots, io_n$ *is* valid *if for all* $1 \leq i \leq n$, *whenever,* $io_i = \mathsf{in}(c_i, u_i)$, *we have that* $R\phi_s{\downarrow} = u_i$ *for some* $R \in \mathcal{T}(\Sigma, \Sigma_0 \cup \mathcal{W} \cup \mathcal{X})$ *where* ϕ_s *is the frame associated to the first-order trace* $io_1 \dots io_i$ (i.e. tr_s *up to the index* i).

Executions, *i.e.* traces of $\mathsf{trace}(P)$, are exactly valid instances of symbolic traces (*i.e.*, valid instances of $\mathsf{trace}_s(P)$).

Lemma 4 *Let P be a bounded protocol. We have that:*

$$\{\mathsf{tr} \mid \mathsf{tr} \textit{ is a ground and valid first-order trace of } P\} \\ = \\ \{\mathsf{tr}\phi\!\downarrow \mid (\mathsf{tr}, \phi) \in \mathsf{trace}(P)\}$$

Proof. The inclusion

$$\{\mathsf{tr}\phi\!\downarrow \mid (\mathsf{tr}, \phi) \in \mathsf{trace}(P)\} \subseteq \{\mathsf{tr}\sigma \mid \mathsf{tr} \in \mathsf{trace}_s(P), \sigma \text{ ground and } \mathsf{tr}\sigma \text{ is valid}\}$$

comes from Lemma 1 which is recalled and proven below. We need to prove that σ is ground, which can be seen in its proof, as θ is ground; and moreover $\mathsf{tr}\sigma$ is valid as $\mathsf{tr}\sigma = \mathsf{tr}\phi\!\downarrow$.

Next, we need to show that $\{\mathsf{tr}\sigma \mid \mathsf{tr} \in \mathsf{trace}_s(P), \sigma \text{ ground and } \mathsf{tr}\sigma \text{ is valid}\} \subseteq \{\mathsf{tr}\phi\!\downarrow \mid (\mathsf{tr}, \phi) \in \mathsf{trace}(P)\}$. Once again a similar induction as the one performed in the proof of Lemma 1 defines a trace $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$, the validity hypothesis ensuring that each transition in the concrete semantics is indeed possible. □

Lemma 1 *Let P be a protocol and $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$. We have that $\mathsf{tr}\phi\!\downarrow = \mathsf{tr}_s\sigma$ for some $\mathsf{tr}_s \in \mathsf{trace}_s(P)$ and some substitution σ.*

Proof. As the symbolic semantics defined in Section 3.2 closely match the semantics described in Section 2.2, as $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ we can define an execution for this trace for the $\rightarrow$ relation which easily corresponds to an execution tr_s for the $\rightarrow_s$ relation. A substitution σ can then be defined inductively on the length of tr: $\sigma_0 = id$ if tr is of length 0, and, assuming we defined σ_n, we extend σ_{n+1} as follows:

- if the next action in tr is $\mathsf{in}(c, R)$ and $\mathsf{in}(c, u)$ in tr_S: there exists a substitution θ such that $R\phi\!\downarrow = u\sigma_n\theta$ as $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$. Then $\sigma_{n+1} = \sigma_n \cup \theta$. Note that because variables are always assumed to be independently renamed, if u binds a variable x, $x \notin dom(\sigma_n)$. We moreover have that $R\phi\!\downarrow = u\sigma_{n+1}$.
- if the next action in tr is $\mathsf{out}(c, \mathsf{w})$ and $\mathsf{out}(c, u)$ in tr_S: then $\sigma_{n+1} = \sigma_n$. $\mathsf{w}\phi = u$ comes directly from the output rule in Section 2.2.
- if the next action in tr is $\mathsf{out}(c, ch_i)$ and $\mathsf{out}(c, ch_i)$ in tr_S: then $\sigma_{n+1} = \sigma_n$, as both rules, concrete and symbolic, are identical.

□

Definition 13 *An algorithm $\mathcal{B}$ is a* reachability blackbox *if it takes as input a first-order trace tr (issued from a bounded protocol P), and returns as output a finite set of substitutions $\sigma_1, ..., \sigma_n$ (with $dom(\sigma_i) \subseteq vars(\mathsf{tr})$) such that:*

- *for each i, the first-order trace $\mathsf{tr}\sigma_i$ is valid; and*
- *if σ is such that $\mathsf{tr}\sigma$ is a valid first-order trace of P then there exists i, and a substitution τ such that* (i) $\mathsf{tr}\sigma = \mathsf{tr}\sigma_i\tau$, *and* (ii) *for every $x \in vars(\mathsf{tr}\sigma_i)$ there exists $R_x \in \mathcal{T}(\Sigma, \Sigma_0 \cup \{\mathsf{w}_1, ..., \mathsf{w}_{\mathsf{ind}_x}\})$ such that $R_x\phi\!\downarrow = x\tau$ where ϕ is the frame associated to $\mathsf{tr}\sigma$ and ind_x is the number of outputs that occur in $\mathsf{tr}_s\sigma_i$ before the first occurrence of an input that contains the variable x.*

All the three decision procedures proposed in [19, 15, 22] are actually reachability blackboxes.

A.2 Our algorithm for trace equivalence

Our algorithm $\mathcal{A}_{\mathcal{B}}$ makes use of a reachability blackbox $\mathcal{B}$. It takes as input two bounded protocols P and Q and returns *yes* when $P \approx Q$; and a minimal (in term of number of actions) witness tr of non-equivalence otherwise.

Our algorithm $\mathcal{A}_{\mathcal{B}}(P, Q)$ 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 $\mathsf{trace}_s(P)$ or $\mathsf{trace}_s(Q)$. Note that since P and Q are bounded, $\mathsf{trace}_s(P)$ and $\mathsf{trace}_s(Q)$ are finite. 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 and Q are trace equivalent.

Iteration steps for level n:

1. Consider every symbolic trace tr_0 in $\mathsf{trace}_s(P)$ of length n and apply $\mathcal{B}$ to it. Consider any substitution σ_0 returned by $\mathcal{B}$. We have that $\mathsf{tr}_1 = \mathsf{tr}_0\sigma_0$ is a valid first-order trace.

2. For any $s, t \in ESt(\mathsf{tr}_1)$ that are unifiable and such that $\mathsf{tr}_1\sigma_1$ is a first-order trace of P where $\sigma_1 = mgu(s, t)$, apply $\mathcal{B}$ to $\mathsf{tr}_2 = \mathsf{tr}_1\sigma_1$. Consider any substitution σ_2 returned by $\mathcal{B}$: $\mathsf{tr}_2\sigma_2$ is a valid first-order trace.

3. Consider a bijective renaming ρ from $vars(\mathsf{tr}_2\sigma_2)$ towards "fresh" public constants. Build a trace $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ such that $\mathsf{tr}\phi\downarrow = (\mathsf{tr}_2\sigma_2)\rho$. Its existence is ensured by Lemma 4

4. Check whether there exists ψ such that $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$. If such a frame does not exist, then return tr. Otherwise, let ψ be a resulting frame.

5. Let K_ϕ (resp. K_ψ) be the subset of $img(\rho)$ of constants occurring in key position in ϕ (resp. ψ). Check whether $K_\psi \subseteq K_\phi$. If there exists $\mathsf{c}_0 \in K_\psi \setminus K_\phi$ then return $\mathsf{tr}\{^{\langle\omega,\omega\rangle}/_{\mathsf{c}_0}\}$. Otherwise, perform step 6.

6. Check whether $\phi \sim \psi$. If the frames are not in static equivalence then return tr. Otherwise, perform steps 1 to 6 by swapping the role of P and Q.

A.3 Termination, soundness, and completeness

Deducibility and static equivalence 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 2 (termination) *Let P and Q be two bounded protocols. The algorithm $\mathcal{A}_{\mathcal{B}}$ applied on P and Q terminates.*

Proof. Termination is ensured by the termination of the blackbox and the decidability of static equivalence. □

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

Proposition 3 (soundness) *Let P and Q be two bounded protocols. If the algorithm $\mathcal{A}_{\mathcal{B}}$ applied on P and Q returns a witness* tr *of non-equivalence, then we have that $P \not\approx Q$.*

Proof. Step 4 clearly returns a witness of non equivalence. At step 5, $\mathcal{A}_{\mathcal{B}}$ returns a trace that is executable in P but which fails in Q since some key becomes non atomic, which yields again a witness of non equivalence. For step 6, note that checking static equivalence for only one resulting frame ψ is actually sufficient thanks to the determinacy hypothesis. □

Establishing completeness is more involved. 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, to capture static equivalence, 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 and Q are non-equivalent (cf. Section A.4).

Proposition 4 (completeness) *Let P and Q be two bounded protocols such that $P \not\approx Q$. The algorithm $\mathcal{A}_{\mathcal{B}}$ applied on P and Q returns a minimal (in term of number of actions) witness* tr *of non-equivalence.*

Proof. (Sketch) Since protocols are determinate, it is sufficient to check static inclusion instead of static equivalence [10]. Static inclusion, denoted $\phi \sqsubseteq_s \psi$, is when ψ satisfies all the equalities of ϕ, and $R\psi\downarrow$ is a message as soon as $R\phi\downarrow$ is a message. So if $P \not\approx Q$, there exists a witness trace tr such that $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ for some ϕ and

1. either $(\mathsf{tr}, \psi) \notin \mathsf{trace}(Q)$ for any ψ;
2. or for every ψ such that $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$, we have that $\phi \not\sqsubseteq_s \psi$.

(or the contrary swapping the role of P and Q.)

In the first case, using Lemma 4, the procedure $\mathcal{B}$ would output a valid trace tr' such that $\mathsf{tr}\phi\downarrow = \mathsf{tr}'\sigma$ for some σ. We can then play tr' in Q and show that if it were a valid trace in Q, tr would also be a valid trace in Q, contradiction. We deduce that $\mathcal{A}_{\mathcal{B}}$ would output tr' (at step 4 or 5), a witness of non-equivalence.

In the second case, following the notation of the previous case, we have that $(\mathsf{tr}', \psi') \in \mathsf{trace}(Q)$ for some ψ' (the choice of the frame is not relevant since they are all in static equivalence due to determinacy of Q). The proof then involves a fine analysis of the relevant equalities that may yield to non static equivalence. We show that whenever there is a witness of non static inclusion for tr (this witness can be an equality test or a test checking whether a given recipe yields a message or not), then there is indeed a trace tr considered at step 2 for which we can exhibit a transformed test that witnesses non static inclusion for tr'. □

The full proof is provided in Section B.2.

A.4 Type-preservation

The specificity of the algorithm we proposed in the previous section is that it further provides a pseudo-well-typed witness whenever the two processes are not in trace equivalence. This can not be achieved using any arbitrary blackbox $\mathcal{B}$. We have to require that the blackbox $\mathcal{B}$ is *type-preserving*.

Definition 14 *A reachability blackbox $\mathcal{B}$ is* type-preserving *if: for any typing system $(\mathcal{T}, \delta)$, for any protocol P type-compliant w.r.t. $(\mathcal{T}, \delta)$, for any well-typed first-order trace tr_s of P given as input, it outputs well-typed substitutions $\sigma_1, \ldots, \sigma_n$ such that:*

$$ESt(\mathsf{tr}_s\sigma_i) \subseteq ESt(\mathsf{tr}_s)\sigma_i \textit{ for any } i \in \{1, \ldots, n\}.$$

Lemma 5 *A type-preserving reachability blackbox exists.*

Most of the existing algorithms (*e.g.* [19, 15, 22]) are actually not type-preserving (since they were not designed with such a goal in mind). However, in the case of the algorithm given in [15], it has already been shown how it can be turned into a type-preserving reachability blackbox [16].

Theorem 3 *Let P and Q be two bounded protocols type-compliant w.r.t. $(\mathcal{T}_1, \delta_1)$ and $(\mathcal{T}_2, \delta_2)$ respectively, and such that $P \not\approx Q$. Assume the algorithm $\mathcal{A}_\mathcal{B}$ uses a type-preserving reachability blackbox $\mathcal{B}$ and a well-typed renaming ρ at step 3. Then $\mathcal{A}_\mathcal{B}(P, Q)$ returns a trace tr such that*

- *either $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ for some ϕ and (tr, ϕ) is pseudo-well-typed w.r.t. $(\mathcal{T}_1, \delta_1)$;*
- *or $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$ for some ψ and (tr, ψ) is pseudo-well-typed w.r.t. $(\mathcal{T}_2, \delta_2)$.*

The proof of Theorem 3 follows from the fact that $\mathcal{A}_\mathcal{B}(P, Q)$ only manipulates well-typed traces. Indeed, it starts from traces provided by $\mathcal{B}$, which are well-typed since $\mathcal{B}$ is type-preserving. Then $\mathcal{A}_\mathcal{B}$ considers only unification between encrypted subterms (which are instances of the encrypted subterms of the protocols). Since P are Q are type-compliant, the resulting traces are well-typed. Actually, $\mathcal{A}_\mathcal{B}$ will output a well-typed trace when a failure occurs at step 4 or at step 6, and a pseudo-well-typed trace when failure occurs at step 5. The complete proof is provided in Section C.

B Proof of Proposition 4

The proof of Proposition 4 requires a number of technicalities so as to reduce a concrete witness of non-equivalence between two protocols into a valid output of the algorithm described in Section A. In particular, recipes used by the attacker to discriminate between two frames need to be modified to be proper tests in the symbolic frames introduced by the said algorithm. Section B.1 will deal with this aspect; while Section B.2 will define a more operational notion of static equivalence and formally link symbolic traces from the algorithm to concrete executions of the protocols, which will be needed to finally prove Proposition 4.

B.1 Simplifying recipes

In this section, we present how equalities between arbitrary recipes can be transformed into a set of equalities between recipes sharing interesting properties, defined in the next definitions. In the following, ϕ and ψ represent two (concrete) frames, while ϕ_S and ψ_S are two symbolic frames such that $\phi = \phi_S\lambda_P$ and $\psi = \psi_S\lambda_Q$, where $\lambda_P = (\theta\phi)\!\downarrow$ and $\lambda_Q = (\theta\psi)\!\downarrow$ and θ is a substitution such that $(vars(\phi_S) \cup vars(\psi_S)) \subseteq dom(\theta)$ and $img(\theta) \subseteq \mathcal{T}_0(\Sigma, \Sigma_0 \cup \mathcal{W})$. These relations are justified later by Lemmas 19 and 21 in Section B.2.

The notions of precompact and compact recipes restrict the tests that can be made by the attacker when trying to distinguish between two frames. Lemma 18 in Section B.2 will prove later this is not, in our setting, an actual restriction.

Definition 15 (precompact recipe) *Given a frame ϕ, a recipe R is said to be* ϕ-precompact *if:*

- *$R\phi\downarrow$ is a message*
- *R contains only destructors*
- *$R\phi\downarrow$ is neither a pair nor an encryption by a key deducible in ϕ*

We now introduce the notion of *symbolic* second-order trace, which is helpful to reason on the objects generated in the decision algorithm of Section A.

Definition 16 *(tr_S, ϕ_S) is a* symbolic *second-order trace of a protocol P if there exists a bijective renaming ρ from $vars(\mathsf{tr}_S\phi_S\downarrow)$ such that $(\mathsf{tr}_S\rho, \phi_S\rho) \in \mathsf{trace}(P)$. In that case, ϕ_S is called a* symbolic *frame.*

Definition 17 (compact recipe) *Given a symbolic frame ϕ_S, R is said to be* ϕ_S-compact *if R is ϕ_S-precompact and $R\phi_S\downarrow$ is not a variable.*

A recipe R is said to be destructor-only if $R \in \mathcal{T}(\Sigma_\mathsf{d}, \Sigma_0 \cup \mathcal{W} \cup \mathcal{X})$, *i.e.* contains no constructor.

We introduce a new predicate on recipe, msg, with the natural semantics: $\phi \models \mathsf{msg}(R)$ if $R\phi\downarrow$ is a message (with or without variables, *i.e.* an element of $\mathcal{T}_0(\Sigma_c, \Sigma_0 \cup \mathcal{N} \cup \mathcal{X})$). Given a term, we introduce a rightmost-first order on its positions, which corresponds to the anti-lexicographic order on positions in a term, denoted by $<$. $\ll$ denotes the order on positions such that $p \ll q$ iff q is a strict prefix of p. Note that: $p \ll q \Rightarrow p < q$. We will sometimes refer to a term as the rightmost term verifying a property, *i.e.* the term verifying this property whose position is the lowest by the $<$ order.

Definition 18 (transformation of concrete recipes for ϕ) *Given ϕ, a destructor-only recipe R, ϕ_S and θ as introduced earlier, we define this transformation $\mathcal{T}$ as follows:*

- *if there exists $R' \in st(R)$ such that $R = C[R']$ and R' is the rightmost recipe verifying one the following two conditions:*
  1. *if $R' = \mathsf{dec}(R_1, R_2)$ and there exists a variable x such that $R_1\phi_S\downarrow = \mathsf{enc}(t, x)$ and $R_2 \neq x$ for some term t, then $\mathcal{T}(R, \mathcal{R}) = (C[\mathsf{dec}(R_1, x)], \mathcal{R} \cup \{\mathsf{msg}(\mathsf{dec}(R_1, x))\})$*
  2. *else, and if there exists a variable y such that $R'\phi_S\downarrow = y$ and $R' \neq y$, then $\mathcal{T}(R, \mathcal{R}) = (C[y\theta]\Downarrow, \mathcal{R} \cup \{R' = y\})$*
- *if no such recipe exists, $\mathcal{T}(R, \mathcal{R}) = (R, \mathcal{R})$;*

where $\Downarrow$ is the normal form associated to the rewriting rules $\pi_i(\langle x_1, x_2\rangle) \to x_i$ and $\mathsf{dec}(\mathsf{enc}(x, y), z) \to x$.

We denote the iterated application of $\mathcal{T}$ to $(R, \emptyset)$ by $\mathcal{T}^*(R)$ or $\mathcal{T}^m(R)$ (when iterated m times).

The iterated transformation $\mathcal{T}^*$ aims at transforming a ϕ-precompact recipe (which is still a quite general class of recipes) into a ϕ_S-compact recipe, *i.e.* a recipe which can reduce properly in a symbolic frame and will satisfy somewhat similar equalities. The next lemmas will gradually prove the properties we need for $\mathcal{T}^*$, ultimately ending with Lemmas 15, 16 and 17.

Lemma 6 (consistency of $\mathcal{R}$) *Let R be a destructor-only recipe, $\mathcal{T}^*(R) = (R^*, \mathcal{R})$: if $R\phi\downarrow$ is a message, $\phi_S \models \mathcal{R}$.*

Proof. A test $\mathsf{msg}(\mathsf{dec}(R, x))$ or $R = x$ is added to $\mathcal{R}$ in Definition 18 only if $R\phi_S\downarrow = \mathsf{enc}(t, x)$ where t is a term (in the first case) or if $R\phi_S\downarrow = x$ (in the second case). As ϕ_S only contains symbolic messages in its image; t has to be a symbolic message. □

This lemma witnesses the fact that the equalities we insert in $\mathcal{R}$, which correspond to equalities holding in the concrete frame, actually hold in its symbolic version. Lemmas 7 and 8 witness rather general properties of destructor-only recipes and of the $\Downarrow$ reduction.

Lemma 7 *If R is a recipe such that $R\phi\downarrow$ is a message, then $R\Downarrow\phi\downarrow = R\phi\downarrow$*

Proof. If $R\phi\downarrow$ is a message, every reduction step of the form $\mathsf{dec}(\mathsf{enc}(x, y), z) \Rightarrow x$ in an innermost derivation in $R\Downarrow$ happens with $y\phi = z\phi$ and is then actually a reduction step of the form $\mathsf{dec}(\mathsf{enc}(x, y), y) \to x$ in $R\phi$. □

Lemma 8 *If R is a destructor-only recipe and ϕ a frame such that $R\phi\downarrow$ is a message, then for every subterm R' of R, $R'\phi\downarrow$ is a message.*

Proof. Suppose there exists a highest (in terms of position, *i.e.* closest to the root) subterm R' of R such that $R'\phi\downarrow$ is not a message and $R = C[R']$ (and C is linear). Let us proceed by induction on C to show that $C[R']\phi\downarrow$ is not a message either. Note that, because R is destructor-only, we only consider C to be destructor-only.

- $C = \_$: then $R = R'$ and $R\phi\downarrow$ is not a message.
- $C = \mathsf{proj}_i(C')$ and $C'[R']\phi\downarrow$ is not a message: then $\mathsf{proj}_i(C'[R'])\phi\downarrow$ will not be a message either.
- $C = \mathsf{dec}(R'', C')$, $C'[R']\phi\downarrow$ and $C'[R']\phi\downarrow$ is not a message. For $C[R']\phi\downarrow$ to be a message, it would require $R''\phi\downarrow = \mathsf{enc}(s, C'[R']\phi\downarrow)$ for some message s. As ϕ only contains messages in its image and $\mathsf{enc}(s, C'[R']\phi\downarrow)$ is not one ($C'[R']\phi\downarrow$ is not a message) the enc function symbol would need to appear in C, which is destructor-only: contradiction. Hence $C[R']\phi\downarrow$ is not a message.
- $C = \mathsf{dec}(C', R'')$, $C'[R']\phi\downarrow$ is not a message: $C[R']\phi\downarrow$ being a message would imply $C'[R']\phi\downarrow = \mathsf{enc}(s, R''\phi\downarrow)$ for some term s. As before, because ϕ contains only messages in its image and $C'[R']\phi\downarrow$ is not, the enc symbol need to occur in $C'[R']$ which is destructor-only, as a subterm of R.

Thus every subterm R' of R is such that $R'\phi\downarrow$ is a message. □

Lemma 9 (preservation of normal forms) *If R is a destructor-only recipe, $(R\theta)\phi\downarrow$ is a message, $(\bar{R}, \mathcal{R}) = \mathcal{T}(R)$, then $(R\theta)\phi\downarrow = (\bar{R}\theta)\phi\downarrow$.*

Proof. Using the notations introduced in Definition 18, $R = C[R']$, R' is the rightmost subterm of R verifying one of the two conditions of $\mathcal{T}$.

1. if $R' = \mathsf{dec}(R_1, R_2)$, $R_1\phi_S\downarrow = \mathsf{enc}(t, x)$ and $R_2 \neq x$: $\bar{R} = C[\mathsf{dec}(R_1, x)]$. We have that $\bar{R}\theta = C[\mathsf{dec}(R_1\theta, x\theta)]\theta$. And thus $(\bar{R}\theta)\phi\downarrow = (C[\mathsf{dec}(R_1\theta, x\theta)]\theta)\phi\downarrow = (C[t\theta]\theta)\phi\downarrow$. As $R_1\phi_S\downarrow = \mathsf{enc}(t, x)$ and $(R\theta)\phi\downarrow$ is a message, $(R'\theta)\phi\downarrow$ is a message too (Lemma 8) and $(R'\theta)\phi\downarrow = (R_1\theta\phi)\downarrow = t\theta\phi$. Hence $(C[t\theta]\theta)\phi\downarrow = (C[R']\theta)\phi\downarrow = (R\theta)\phi\downarrow$, giving the result.

2. else, and if $R'\phi_S\downarrow = y$ and $R' \neq y$: $\bar{R} = C[y\theta]\Downarrow$. We have $y\theta\phi\downarrow = (R'\theta)\phi\downarrow$. Indeed, as $R'\phi_S\downarrow = y$, $R'\phi_S\downarrow\theta\phi\downarrow = y\theta\phi\downarrow$. Then $(R'\theta)\phi\downarrow = y\theta\phi\downarrow$, as $\phi = \phi_S\lambda_P$ (works with $\psi = \psi_S\lambda_Q$ too, thanks to Lemma 21). To conclude, we show that $(R\theta)\phi\downarrow = (C[(R'\theta)\phi\downarrow]\theta)\phi\downarrow$, and as $(R\theta)\phi\downarrow$ is a message and by Lemma s7, this is equal to $(C[y\theta\phi\downarrow]\theta)\phi\downarrow = (C[y\theta]\theta)\phi\downarrow = (C[y\theta]\Downarrow\theta)\phi\downarrow = (\bar{R}\theta)\phi\downarrow$.

□

This lemma proves that the transformation $\mathcal{T}$, from the point of view of the concrete frame, does not alter the normal form of tests (up to a particular substitution θ). Previous lemmas are stated with ϕ and ϕ_S, but can be symmetrically applied with ψ and ψ_S. When the context is not obvious, we will denote by $\mathcal{T}_\phi$ the transformation introduced at Definition 18 when applied with ϕ_S, and $\mathcal{T}_\psi$ when applied with ψ_S.

Lemma 10 (termination of $\mathcal{T}^*$) *$\mathcal{T}$ is deterministic and $\mathcal{T}^*(R)$ is well-defined.*

Proof. Introduce an ordering on variables based on how soon they appear in the trace (tr_S, ϕ_S) and consider the induced multi-set order $<_{\mathsf{var}}$. The second item in Definition 18 strictly reduces this measure. Now consider the number of destructor of a (destructor-only) recipe R, plus the number of variables in $dom(\phi)$ (without counting the variables in $dom(\theta)$). Let $<_{\mathsf{size}}$ be the order induced by this measure. The first item in Definition 18 strictly reduces this measure. Finally, let $<$ be the lexicographical order built on $(<_{\mathsf{var}}, <_{\mathsf{size}})$. The transformation $\mathcal{T}$ decreases its induced measure. □

Lemmas 11, 12 and Corollary 2 provide the general invariants for the transformation $\mathcal{T}$: mostly that it operates locally, does not introduce new constructors and preserves the fact for a subterm of being a message.

Lemma 11 *If R is ϕ-precompact, then for every $n \in \mathbb{N}$, $\mathcal{T}_\phi^n(R)$ is destructor-only.*

Proof. Suppose there exists $n_0 \in \mathbb{N}$ such that $R_{n_0} = \mathcal{T}^{n_0}(R)$ contains a constructor c at position p. Let us further assume p is the highest position where a constructor occurs.

- if $p = \epsilon$: as $(R_{n_0}\theta)\phi\downarrow = R\phi\downarrow$ by Lemma 9, R cannot be ϕ-precompact,
- if $p > \epsilon$ and $c = \langle\_,\_\rangle$: c occurs below a destructor d (as p is the highest position a constructor can appear).
  - If $d = \mathsf{proj}_i$ for $i \in \{1, 2\}$: as R is destructor-only, c is introduced by a replacement $C[y\theta]\Downarrow$ and because $\mathsf{proj}_i(\langle x_1, x_2\rangle) \Rightarrow x_i$, c would have been reduced.
  - If $d = \mathsf{dec}$ and c is in plaintext position of d: if c is not reduced, $(R_{n_0}\theta)\phi\downarrow$ is not a message, then $(R_{n_0}\theta)\phi\downarrow = R\phi\downarrow$, by Lemma 9, implies $R\phi\downarrow$ is not a message either, and thus not ϕ-precompact.
  - Else, if $d = \mathsf{dec}$ and c is in key position of d: the key is not atomic, and similarly $R\phi\downarrow$ is not a message, and thus not ϕ-precompact.
- if $p > \epsilon$ and $c = \mathsf{enc}(\_,\_)$: the same reasoning as the previous case can be applied (interverting the first two subcases).

Hence R_{n_0} cannot contain constructors if R is ϕ-precompact. □

Lemma 12 *Let R^{init} be a destructor-only recipe and $R = \mathcal{T}_\phi^n(R^{init})$. If $R = C[R']_p$, R is destructor-only, $(R'\theta)\phi\downarrow$ is a message and p is lesser w.r.t. $<$ than the next position where $\mathcal{T}_\phi$ is applied on R, then $R|_q\phi_S\downarrow$ is a message for any $q \ll p$ or $q = p$.*

Proof. We prove by induction that $R|_q\phi_S{\downarrow}$ is a message.

- if $R|_q = \mathsf{w}$, $\mathsf{w}\phi_S$ is a message,
- if $R|_q = x$, $x\phi_S = x$ is a message,
- if $R|_q = \mathsf{enc}(R_1, R_2)$ or $R|_q = \langle R_1, R_2\rangle$, $R_1\phi_S{\downarrow}$ and $R_2\phi_S{\downarrow}$ are messages by induction hypothesis, then $R|_q\phi_S{\downarrow}$ is a message too,
- if $R|_q = \mathsf{proj}_i(R'')$ and $R''\phi_S{\downarrow}$ is a message:
  - if $R''\phi_S{\downarrow} = x$ for some variable x: a variable in R'' can only be introduced in key position of a dec, not a proj_i, impossible.
  - if $R''\phi_S{\downarrow} = \langle R_1, R_2\rangle$: then $\mathsf{proj}_i(\langle R_1, R_2\rangle)\phi_S{\downarrow}$ is a message,
  - if $R''\phi_S{\downarrow} = \mathsf{enc}(u, v)$: as $(R''\theta)(\phi_S\lambda_P){\downarrow} = (R''\theta)\phi{\downarrow}$, there exist u', v' two terms such that $(R''\theta)\phi{\downarrow} = \mathsf{enc}(u', v')$. Thus $(\mathsf{proj}_i(R'')\theta)\phi{\downarrow}$ is not a message, and $(R'\theta)\phi{\downarrow}$ is not a message either, by Lemma 8 as R is destructor-only. Contradiction.
- if $R|_q = \mathsf{dec}(R_1, R_2)$, $R_1\phi_S{\downarrow}$ and $R_2\phi_S{\downarrow}$ are messages:
  - if $R_1\phi_S{\downarrow} = x$ for some variable x: a variable can only occur in a key position, impossible. Either $R_1 = x$, which can only occur in a key position, impossible; or $R_1 \neq x$ and $R_1\phi_S{\downarrow} = x$, in which case $\mathcal{T}$ would be applied at position $q.1 < p$
  - if $R_1\phi_S{\downarrow} = \langle u, v\rangle$: see the third point of the previous case.
  - if $R_1\phi_S{\downarrow} = \mathsf{enc}(u, x)$ for some term u and some variable x. By Definition 18, as $\mathcal{T}$ was applied at position q, $R_2 = x$ and $\mathsf{dec}(R_1, x)\phi_S{\downarrow}$ is a message.
  - if $R_1\phi_S{\downarrow} = \mathsf{enc}(u, k)$ for some term u and some non-variable atom k: then $R_2\phi_S{\downarrow} \neq x$ for any variable x, as R_2 is a position lesser than p. Thus $R_2\phi_S{\downarrow} = k'$ for some atom k', and $R_2\phi{\downarrow} = k'$ as k' is not a variable. Similarly $R_1\phi{\downarrow} = \mathsf{enc}(u', k)$ for some term u'. $(R\theta)\phi{\downarrow}$ is a message implies (as R is destructor-only and by Lemma 8) $k = k'$ and that $\mathsf{dec}(R_1, R_2)\phi_S{\downarrow}$ is a message too.

□

Corollary 2 *Let R^{init} be a destructor-only recipe and $R = \mathcal{T}_\phi^n(R^{init})$. If $R = C[R']_p$, R is destructor-only, $(R'\theta)\phi{\downarrow}$ is a message and $\mathcal{T}_\phi(R) = (R, \mathcal{R})$ then $R|_q\phi_S{\downarrow}$ is a message for any $q \ll p$ or $q = p$.*

Proof. Same proof as Lemma 12, except we invoke the fact the transformation does not alter R any more in cases where we derive an impossibility. □

The following lemma intends to show that equalities between transformed test $\mathcal{T}^*(R)$ actually correspond to the unification performed by the algorithm at step 2. Indeed, the normal forms of ϕ_S-compact recipes are encrypted subterms that can be unified in this step.

Lemma 13 (compacification effect of $\mathcal{T}^*$) *With previous notations, if $\mathcal{T}^*(R) = (R^*, \mathcal{R})$: if R is ϕ-precompact then R^* is ϕ_S-compact.*

Proof. As R is ϕ-precompact, by Lemma 11, any (iterated) application of $\mathcal{T}_\phi$ to R yields a destructor-only recipe. We then use Lemma 12 repeatedly and finally Corollary 2 with $p = \epsilon$ (the root position) and for $q = p$ to conclude $R^*\phi_S\downarrow$ is a message. Invoking Lemma 9 at each step finally gets us $R\phi\downarrow = (R^*\theta)\phi\downarrow$. As $(R^*\theta)\phi\downarrow = R^*\phi_S\downarrow\lambda_P$, we can conclude that $R^*\phi_S\downarrow$ is not a pair nor an encryption with a deducible key (for the latter, consider the case where $R^*\phi_S\downarrow = \mathsf{enc}(u_1, u_2)$: as R is ϕ-precompact, $u_2\lambda_P$ is not deducible in ϕ; but if u_2 were deducible in ϕ_S, $u_2\lambda_P$ would be in $\phi \cup \lambda_P$, and thus in ϕ. Contradiction). □

Lemma 14 (uniformity of $\mathcal{T}$) *Let ϕ, ϕ_S, ψ, ψ_S be as introduced, R be a destructor-only recipe, $\mathcal{T}^*_\phi(R) = (R^*_1, \mathcal{R}_1)$ and $\mathcal{T}^*_\psi(R) = (R^*_2, \mathcal{R}_2)$. If $\psi_S \models \mathcal{R}_1$ and $\phi_S \models \mathcal{R}_2$, then $R^*_1 = R^*_2$.*

Proof. Let $(R^k_1, \mathcal{R}^k_1) = \mathcal{T}^k_\phi(R)$ and $(R^k_2, \mathcal{R}^k_2) = \mathcal{T}^k_\psi(R)$ be the iterated application of $\mathcal{T}$ with both symbolic frames. We will prove inductively that $R^k_1 = R^k_2$ and $\mathcal{R}^k_1 = \mathcal{R}^k_2$.

- for $k = 0$, the initial recipe, R, is identical in both cases and $\mathcal{R}^0_1 = \emptyset = \mathcal{R}^0_2$.
- Let us assume we obtained the result up to some k: $R^k_1 = R^k_2 = R^k$. Note that $\mathcal{R}^k_i \subseteq \mathcal{R}_i$ for $i \in \{1, 2\}$ implies $\phi_S \models \mathcal{R}^k_2$ and $\psi_S \models \mathcal{R}^k_1$. Suppose now that $R^k = C[R']|_p$ and p is the lowest position w.r.t. $<$ such that any of the two rules in Definition 18 can apply with *either* ϕ_S or ψ_S. For instance, suppose it is true for ϕ_S:

  1. if $R' = \mathsf{dec}(R_1, R_2)$, there exist a variable x and a term t such that $R_1\phi_S\downarrow = \mathsf{enc}(t, x)$, $R_2 \neq x$, then $R^{k+1}_1 = C[\mathsf{dec}(R_1, x)]_p$ and $\mathcal{R}^{k+1}_1 = \mathcal{R}^k \cup \{\mathsf{msg}(\mathsf{dec}(R_1, x))\}$. As $\psi_S \models \mathcal{R}_1$ and $\mathcal{R}^{k+1}_1 \subseteq \mathcal{R}_1$, $\psi_S \models \mathsf{msg}(\mathsf{dec}(R_1, x))$. Hence $R_1\psi_S\downarrow = \mathsf{enc}(s, x)$ for some term s and the same rule of $\mathcal{T}_\psi$ can be applied at the same position, and will, as p is the lowest position where a rule of $\mathcal{T}$ is applicable for both symbolic frames, and then: $R^{k+1}_2 = R^{k+1}_1$ and $\mathcal{R}^{k+1}_2 = \mathcal{R}^{k+1}_1$.
  2. If there exists a variable y such that $R'\phi_S\downarrow = y$ and $R' \neq y$, then $R^{k+1}_1 = C[y\theta]\Downarrow$ and $\mathcal{R}^{k+1}_1 = \mathcal{R}^k \cup \{R' = y\}$. As $\psi_S \models \mathcal{R}_1$ and $\mathcal{R}^{k+1}_1 \subseteq \mathcal{R}_1$, $\psi_S \models R' = y$, *i.e.* $R'\psi_S\downarrow = y$. The same rule of $\mathcal{T}_\psi$ can thus be applied at the same position, and will, as p is the lowest position where a rule of $\mathcal{T}$ is applicable for both symbolic frames, and then: $R^{k+1}_2 = R^{k+1}_1$ and $\mathcal{R}^{k+1}_2 = \mathcal{R}^{k+1}_1$.

  The case where p corresponds to a position *w.r.t.* ψ_S is handled symmetrically.

Applying that result for n such that $\mathcal{T}^*_\phi(R) = \mathcal{T}^n_\phi(R)$ leads to the final result. □

The next two lemmas finally ensure $\mathcal{T}^*$ does not alter the equalities in any nefarious way. Together they demonstrate that if a test holds in a concrete concrete frame, the transformed equality, up to a unification performed at step 2 of the algorithm, will hold in the symbolic frame.

Lemma 15 (soundness of $\mathcal{T}^*$) *Let R_1, R_2 be two destructor-only recipes, ψ, ψ_S, λ_Q be as expected, $(R^*_1, \mathcal{R}_1) = \mathcal{T}^*_\psi(R_1)$ and $(R^*_2, \mathcal{R}_2) = \mathcal{T}^*_\psi(R_2)$. Then, for any $i \in \{1, 2\}$, $R_i\psi\downarrow = R^*_i\psi_S\lambda_Q\downarrow$; and thus $R^*_1\psi_S\downarrow = R^*_2\psi_S\downarrow$ implies $R_1\psi\downarrow = R_2\psi\downarrow$.*

Proof. Iteration of Lemma 9 gives $R_i\psi\downarrow = (R^*_i\theta)\psi\downarrow$; and we assumed $\psi = \psi_S\lambda_Q$ with $\lambda_Q = (\theta\psi)\downarrow$. Hence, $R^*_i\psi_S\lambda_Q\downarrow = (R^*_i\theta)\psi_S(\theta\psi)\downarrow = (R^*_i\theta)\psi\downarrow = R_i\psi\downarrow$. □

Lemma 16 (completeness of $\mathcal{T}^*$) *Let R_1, R_2 be two ϕ-precompact recipes, ϕ, ϕ_S, λ_P be as expected, $(R_1^*, \mathcal{R}_1) = \mathcal{T}_\phi^*(R_1)$ and $(R_2^*, \mathcal{R}_2) = \mathcal{T}_\phi^*(R_2)$. Then: $R_1\phi\downarrow = R_2\phi\downarrow$ implies there exists σ which is a mgu of two ϕ_S-compact recipes in ϕ_S such that $R_1^*\phi_S\sigma\downarrow = R_2^*\phi_S\sigma\downarrow$.*

Proof. Iteration of Lemma 9 gives $R_i\phi\downarrow = R_i^*\phi\downarrow$; and we assumed that $\phi = \phi_S\lambda_P$. Then, $R_1\phi\downarrow = R_2\phi\downarrow$ is the same as $R_1^*\phi_S\lambda_P\downarrow = R_2^*\phi_S\lambda_P\downarrow$. Thus $\sigma = mgu(R_1^*\phi_S, R_2^*\phi_S) \neq \bot$. As R_1^* and R_2^* are ϕ_S-compact (Lemma 13), we are done. □

Unfortunately, the transformation $\mathcal{T}$ may, in some cases, transform a recipe reducing to a term which was not a message into a new recipe reducing to a symbolic message. The next lemma ensures these special case can be handled in the main proof of completeness of the algorithm.

Lemma 17 (preservation of messages) *Let ϕ, ϕ_S, ψ, ψ_S be as expected, R a ϕ-precompact recipe such that $\mathcal{T}_\phi^*(R) = (R^*, \mathcal{R}_1)$, $\psi_S \models \mathcal{R}_1$, $\mathcal{T}_\psi^*(R) = (R^*, \mathcal{R}_2)$ and $\phi_S \models \mathcal{R}_2$. If $R\phi\downarrow$, $R^*\phi_S\downarrow$, $R^*\psi_S\downarrow$ are messages but $R\psi\downarrow$ is not, then there either exist R_1^0 and R_2^0 two ϕ-precompact recipes such that $R_1^0\phi\downarrow = R_2^0\phi\downarrow$ and $R_1^0\psi\downarrow \neq R_2^0\psi\downarrow$ or there exists a ϕ_S-compact recipe R_0 such that $R_0\phi_S\downarrow$ is a message but $R_0\psi_S\downarrow$ is not.*

Proof. Let p be the minimal position in R of a destructor d which is not reduced in $R\psi\downarrow$. Suppose $\mathcal{T}^*(R) = \mathcal{T}^n(R)$ (note that Lemma 14 guarantees that $\mathcal{T}_\psi = \mathcal{T}_\phi$ when applied on R). Let R_i be a shortcut for the first argument of $\mathcal{T}^i(R)$. Let $q_1, q_2, \ldots, q_n$ be the successive positions where $\mathcal{T}$ is applied; and $i_0 \in \{1, \ldots, n\}$ such that $\forall 0 \leq j < i_0, (R_j\theta)\psi\downarrow$ contains d at position p and $(R_{i_0}\theta)\psi\downarrow$ does not. Let $\mathcal{P}$ be the set of positions in R. We define a partition of $\mathcal{P}$ as follows:

$$\mathcal{P} = \{p\} \cup \mathcal{P}_1 \cup \mathcal{P}_2 \cup \mathcal{P}_3 \cup \mathcal{P}_4 \cup \mathcal{P}_5$$

where:

- $\mathcal{P}_1$ is the set of positions on the "plaintext line" of p (always-leftmost children of p),
- $\mathcal{P}_2$ is the set of positions q such that $q < p$ and $q \notin \mathcal{P}_1$,
- $\mathcal{P}_3$ is the set of strict parents of p,
- $\mathcal{P}_4$ is the set of always-leftmost children of elements of $\mathcal{P}_3$ which are not in $\mathcal{P}_3 \cup \mathcal{P}_1 \cup \{p\}$,
- $\mathcal{P}_5$ is the set of positions greater than p w.r.t. $<$ which are not in $\mathcal{P}_3$ nor $\mathcal{P}_4$.

The core argument of this proof relies on a disjunction on the nature of q_{i_0}:

- if $q_{i_0} \in \mathcal{P}_2 \cup \mathcal{P}_5$: impossible, as any transformation by $\mathcal{T}$ at these positions cannot affect anything at position p. Indeed all these elements appears below a key position, *i.e.* a position which is the key of some dec.
- $q_{i_0} = p$: impossible, as if $\mathcal{T}$ were applied at this position, it would require $R_{i_0-1}|_p\psi_S\downarrow$ to be some variable x, and thus $(R_{i_0-1}\theta)\psi\downarrow$ would not contain d (as $\psi = \psi_S\lambda_Q$).

- $q_{i_0} \in \mathcal{P}_1$: by definition of i_0, $(R_{i_0-1}\theta)\psi\downarrow$ contains d at position p while $(R_{i_0}\theta)\psi\downarrow$ does not and $q_{i_0} < p$ (by definition of $\mathcal{P}_1$). Let $R_{i_0-1} = C[R']_{q_{i_0}}$. Because $\mathcal{T}$ is applied at q_{i_0} and d deleted at this step, necessarily $R'\psi_S\downarrow = x$ for some variable x and $R_{i_0} = C[x\theta]_{q_{i_0}}\Downarrow$; and d is deleted through the $\Downarrow$-reduction. Two cases can occur *a priori*:

  - $d = \mathsf{proj}_i$. In that case, the same reductions appear with ϕ: hence $x\theta$ has to contain the pair which is deleted by d, and as recipes are the same no matter the frame we consider, the same reduction could occur in ψ, hence d would not occur in $R\psi\downarrow$ (We use Lemma 9 to get the preservation of normal forms through the process). Impossibility.
  - $d = \mathsf{dec}$. Then there exists a recipe in the key position of d at position $p.2 < p$: $R_2'^0$. Because $(R_2'^0\theta)\psi\downarrow$ is a message (minimality of p) and Lemma 12 (third item, $p.2 < q_{i_0}$), $R_2'^0\psi_S\downarrow$ is a message. As R is ϕ-precompact and by Lemma 11, $R_2'^0$ is destructor-only. Note that $R_2'^0$ may contain variables in key positions, due to the first item of Definition 18. Because keys are atomic, for each variable in key position x, $x\lambda_P$ and $x\lambda_Q$ are atomic, thus there exist destructor-only recipe R_x for each of them. So we define $R_2^0 = R_2'^0[R_x/x]$. R_2^0 is now ϕ-precompact, $(R_2'^0\theta)\psi\downarrow = R_2^0\downarrow\psi$ and $(R_2'^0\theta)\phi\downarrow = R_2^0\downarrow\phi$. As $\Downarrow$-reduction only occurs at recipe level, there also exists a recipe R_1^0 for the encryption key used by the constructor reduced by d. Because keys are atomic, we also can always assume R_1^0 to be ϕ-precompact in this setting. We now show that $R_1^0\psi\downarrow \neq R_2^0\psi\downarrow$ and $R_1^0\phi\downarrow = R_2^0\phi\downarrow$. These equalities are derived from the fact that $R\psi\downarrow$ still contains d but $R\phi\downarrow$ is a message. Hence we obtain two recipes as in the statement of the lemma.

- $q_{i_0} \in \mathcal{P}_3$: then $R_{i_0-1}|_{q_{i_0}.1}\psi_S\downarrow = \mathsf{enc}(t, x)$ for some term t and variable x, and $p \ll q_{i_0}.2$ (or else $R_{i_0-1}|_{q_{i_0}.1}\psi_S\downarrow$ would not reduce if it included d). We get that $R_{i_0} = C[\mathsf{dec}(R_{i_0-1}|_{q_{i_0}.1}, x)]_{q_{i_0}}$. Let $R_0 = R_{i_0-1}|_{q_{i_0}.2}$. In particular, p is a position of R_0. As $q_{i_0}.2 < q_{i_0}$, R_0 is ϕ_S-compact (Lemmas 11 and 12). We now want to show that $R_0\psi_S\downarrow$ is not a message: as $(R_0\theta)\psi\downarrow$ contains d by definition, if $R_0\psi_S\downarrow$ did not, $R_0\psi_S\downarrow\lambda_Q\downarrow = (R_0\theta)\psi\downarrow$ would not either, which contradicts our hypothesis. Hence we get our recipe R_0 as in the second case of the statement of the lemma.

- $q_{i_0} \in \mathcal{P}_4$: we apply the same reasoning as if $q_{i_0} \in \mathcal{P}_3$, except for the fact it is now the second rule of Definition 18 which is applied at q_{j_0}, where q_{j_0} the longest common prefix of q_{i_0} and p, and necessarily $R_{i_0-1}|_{q_{j_0}} = \mathsf{dec}(R_l, R_r)$. If it were a proj_i, j_0 would not be the longest common prefix.

Thus the result for any position q_{i_0}, and the existence of such recipes in general. □

B.2 Decision for bounded protocols

Here we detail how symbolic traces can be formally linked to the concrete executions of the protocol so as to properly prove Proposition 4.

First, rather than deal with the usual notion of static equivalence or inclusion (*i.e.* only the direct implications in Definition 2), we use a variation which we prove to be equivalent in the next lemma.

Lemma 18 (alternative definition of static inclusion) *We say that $\phi_1 \sqsubseteq' \phi_2$ if:*

- *for every ϕ_1-precompact recipe M, if $M\phi_1\downarrow$ is an atom, then $M\phi_2\downarrow$ is an atom,*
- *for every ϕ_1-precompact recipe M, $M\phi_2\downarrow$ is a message,*
- *for every ϕ_1-precompact recipe M and N, if $M\phi_1\downarrow = N\phi_1\downarrow$ then $M\phi_2\downarrow = N\phi_2\downarrow$.*

and similarly, $\phi_1 \sim' \phi_2$ if $\phi_1 \sqsubseteq' \phi_2$ and $\phi_2 \sqsubseteq' \phi_1$.
Then $\phi_S \sqsubseteq \phi_2$ if, and only if, $\phi_1 \sqsubseteq' \phi_2$.

Proof. We need to show that $\phi_1 \sqsubseteq \phi_2$ if, and only if, $\phi_1 \sqsubseteq' \phi_2$, where $\sqsubseteq$ denotes the static inclusion, *i.e.* only the direct implications in Definition 2. The implication $\phi_1 \sqsubseteq \phi_2 \Rightarrow \phi_1 \sqsubseteq' \phi_2$ is direct, as this definition only examine fewer tests than the original one and the ability for the attacker to test whether a recipe R reduces to an atom is already ensured by tests of the form $\mathsf{dec}(\mathsf{enc}(\mathsf{w}_1, R)) = \mathsf{w}_1$. So let us consider the converse implication, and assume that $\phi_1 \sqsubseteq' \phi_2$ and suppose $\phi_1 \not\sqsubseteq \phi_2$. We proceed by induction on recipes, proving our transformation of a single recipe or a pair of recipes strictly decreases the number of constructors (in the single recipe or the sum for pairs). We claim that if we have a witness of static non-inclusion, there is a recipe or a pair of recipes which are destructor-only witnessing that. In the following, M will denote a recipe such that $M\phi_1\downarrow$ is a message but $M\phi_2\downarrow$ is not; M_1 and M_2 will denote two recipes such that $M_i\phi_j\downarrow$ is a message for $i, j \in \{1, 2\}$, $M_1\phi_1\downarrow = M_2\phi_2\downarrow$ but $M_1\phi_2\downarrow \neq M_2\phi_2\downarrow$; and C will be a linear destructor-only context. Let n be the number of constructors in M or the sum of the number of constructors in M_1 and M_2.

- If $n = 0$: M (resp. M_1 and M_2) is destructor-only,
- M contains pairing:
  1. $M = C[\mathsf{dec}(\langle M_1^0, M_2^0\rangle, M_3^0)]$: impossible, as $M\phi_1\downarrow$ would not be a message,
  2. $M = C[\mathsf{proj}_i(\langle M_1^0, M_2^0\rangle)]$: then $M' = C[M_i^0]$ has strictly less than n constructors, is a message in ϕ_1 but still not in ϕ_2,
  3. $M = \langle M_1^0, M_2^0\rangle$: then there exists $i \in \{1, 2\}$ such that M_i^0 is a message in ϕ_1 while not in ϕ_2; and M_i^0 contains strictly less than n constructors,
- M contains encryption:
  1. $M = \mathsf{enc}(M_1^0, M_2^0)$: three subcases must be examined:
     (a) if $M_2^0\phi_2\downarrow$ is a message and is *not* an atom: then M_2^0 contains strictly less than n constructors; and $M_2^0\phi_1\downarrow$ is an atom (as M reduces to a message in ϕ_1),
     (b) else, if $M_2^0\phi_2\downarrow$ is not a message: M_2^0 contains strictly less than that n constructors,
     (c) else, if $M_2^0\phi_2\downarrow$ is an atom: then M_1^0 is a message in ϕ_1 but not in ϕ_2 and contains strictly less than n constructors,
  2. $M = C[\mathsf{proj}_i(\mathsf{enc}(M_1^0, M_2^0))]$: impossible, as $M\phi_1\downarrow$ would not be a message,

3. $M = C[\mathsf{dec}(\mathsf{enc}(M_1^0, M_2^0), M_3^0)]$: in particular, $M_2^0\phi_1{\downarrow} = M_3^0\phi_1{\downarrow}$. If M_2^0 and M_3^0 are both messages in ϕ_2 and $M_2^0\phi_2{\downarrow} \neq M_2^0\phi_2{\downarrow}$: the pair (M_2^0, M_3^0) contains $n-1$ constructors and is a witness of non-inclusion. Else, if $M_i^0\phi_2{\downarrow}$ is not a message for some $i \in \{2,3\}$, then it contains strictly less than n constructors. In the remaining case, $M_1^0\phi_2{\downarrow}$ is not a message, and contains strictly less than n constructors,

- M_1 contains pairing (or symmetrically, M_2 contains pairing), using the same numbering as the case where M did contain pairing:

  1. this case cannot happen,
  2. $C[M_i^0]\phi_j{\downarrow} = M_1\phi_j{\downarrow}$ for $j \in \{1,2\}$, and $C[M_i^0]$ contains strictly less than n constructors,
  3. $M_i^0\phi_2{\downarrow} = \mathsf{proj}_i(M_1)\phi_2{\downarrow}$ and $M_1\phi_2{\downarrow} \neq M_2\phi_2{\downarrow}$ implies that either $\mathsf{proj}_i(M_2)\phi_2{\downarrow}$ is not a message, or $M_i^0\phi_2{\downarrow} \neq \mathsf{proj}_i(M_2)\phi_2{\downarrow}$; which in both cases counts strictly less than n constructors,

- M_1 contains encryption (or symmetrically, M_2 contains encryption), using the same numbering as the case where M did contain encryption:

  1. $M_2^0\phi_1{\downarrow}$ and $M_2^0\phi_2{\downarrow}$ are both messages and atom and $M_1\phi_2{\downarrow} \neq M_2\phi_2{\downarrow}$ implies that either $\mathsf{dec}(M_2, M_2^0)\phi_2{\downarrow}$ is not a message or that $M_1^0\phi_2{\downarrow} \neq \mathsf{dec}(M_2, M_2^0)\phi_2{\downarrow}$; which in both cases counts strictly less than n constructors,
  2. this case cannot happen,
  3. $M_1\phi_2{\downarrow}$ is a message implies $M_2^0\phi_2{\downarrow} = M_3^0\phi_2{\downarrow}$ and $M_1^0\phi_2{\downarrow}$ is a message. Then M_1^0 contains strictly less constructors than M_1, and we get $M_1^0\phi_1{\downarrow} = M_2\phi_1{\downarrow}$ while $M_1^0\phi_2{\downarrow} \neq M_2\phi_2{\downarrow}$.

At this point, we proved that we can only consider destructor-only witnesses of $\phi_1 \not\sqsubseteq \phi_2$. Suppose now M, M_1 and M_2 are destructor-only: we need to prove they are ϕ_1-precompact, *i.e.* we show they do not reduce to a pair or an encryption with a deducible key.

- if $M\phi_1{\downarrow} = \langle s, t\rangle$: there exists $i \in \{1,2\}$ such that $\mathsf{proj}_i(M)$ is still a message in ϕ_1 but not in ϕ_2,

- if $M\phi_1{\downarrow} = \mathsf{enc}(s,k)$ and k is deducible in ϕ_1, which implies there exists a destructor-only recipe R such that $R\phi_1{\downarrow} = k$ (consider the $\Downarrow$ normalisation of any recipe reducing to k). $\mathsf{dec}(M,R)$ is a message in ϕ_1 while not in ϕ_2,

- if $M_1\phi_1{\downarrow} = \langle s,t\rangle = M_2\phi_1{\downarrow}$: there exists $i \in \{1,2\}$ such that $\mathsf{proj}_i(M_1)\phi_1{\downarrow} = \mathsf{proj}_i(M_2)\phi_1{\downarrow}$ but $\mathsf{proj}_i(M_1)\phi_2{\downarrow} \neq \mathsf{proj}_i(M_2)\phi_2{\downarrow}$ or $\mathsf{proj}_i(M_j)\phi_2{\downarrow}$ is not a message for some $j \in \{1,2\}$.

- if $M_1\phi_1{\downarrow} = \mathsf{enc}(s,k) = M_2\phi_1{\downarrow}$, and k is deducible: as previously, there exists a destructor-only recipe R such that $R\phi_1{\downarrow} = k$. In that case $\mathsf{dec}(M_1,R)\phi_1{\downarrow} = \mathsf{dec}(M_2,R)\phi_1{\downarrow}$. Then either $\mathsf{dec}(M_i,R)\phi_2{\downarrow}$ is not a message for some $i \in \{1,2\}$ or $\mathsf{dec}(M_1,R)\phi_2{\downarrow} \neq \mathsf{dec}(M_2,R)\phi_2{\downarrow}$.

Note that this last transformation does not introduce any constructor and strictly decreases the normal forms of M, M_1 and M_2 in ϕ_1. Hence, if $\phi_1 \not\sqsubseteq \phi_2$, then $\phi_1 \not\sqsubseteq' \phi_2$. □

Lemmas 19 and 21 provide the link between any concrete second-order trace, as the attacker can build, and the second-order traces which are generated at step 3 of our algorithm. They also provide the relations between concrete and symbolic frames we needed at the beginning of Section B.1.

Step 3 in the algorithm induces a renaming ρ. As this renaming is bijective, and only meant to provide concrete traces of P and Q, in the following statements and proofs, we will omit it and refer to the non-renamed trace in the algorithm by (tr_S, ϕ_S), which will then by symbolic.

Lemma 19 (existence of a symbolic trace) *For any $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$, there exists a* symbolic *second-order trace (tr_S, ϕ_S) of P generated by $\mathcal{A}_{\mathcal{B}}(P, Q)$, a first-order substitution λ_P and a second-order substitution θ such that $\mathsf{tr}\phi\downarrow = \mathsf{tr}_S\phi_S\lambda_P\downarrow$ (in particular: $\phi = \phi_S\lambda_P$), $\mathsf{tr}_S\theta\phi\downarrow = \mathsf{tr}\phi\downarrow$ (the first-order input terms are identical), and such that for every $x \mapsto R_x \in \theta$, R_x is built from the initial knowledge of the attacker and the outputs which preceded the introduction of x in tr_S, $R_x\phi_S\downarrow$ is a message and for every variable x occurring in key position in $\mathsf{tr}_S\phi_S\downarrow$, $R_x\phi_S\downarrow$ is an atom,*

Proof. For (tr, ϕ) we can derive the existence of $\mathsf{tr}_0 \in \mathsf{trace}_s(P)$ and a substitution σ such that $\mathsf{tr}\phi\downarrow = \mathsf{tr}_0\sigma$ (Lemma 1). Using Definition 13 we get there exists a first-order substitution σ_1 generated by $\mathcal{B}$ applied to tr_0 and a first-order substitution τ such that $\mathsf{tr}_0\sigma = \mathsf{tr}_1\tau$, where $\mathsf{tr}_1 = \mathsf{tr}_0\sigma_1$. (tr_S, ϕ_S) is obtained by lifting tr_1 to second-order with arbitrary valid recipes (*i.e* progressively constructible by the attacker) and storing the outputs of tr_1 in ϕ_S. A fortiori then, $\phi_S\tau = \phi$. Let $\lambda_P = \tau$. To define θ we choose for every $x \in vars(\mathsf{tr}_S)$ a recipe R_x of $x\lambda_P$, then $\theta = \{x \mapsto R_x \text{ for } x \in vars(\mathsf{tr}_S)\}$. Finding such a recipe is always possible thanks to Definition 13. We can moreover assume $R_x\phi_S\downarrow$ is a symbolic message and if x appears in key position inside $\mathsf{tr}_S\phi_S\downarrow$, then $x\theta\phi_S\downarrow$ is an atom. Indeed, let us order variables in tr_S with their order of apparition (variables introduced simultaneously in an input can be ordered arbitrarily) in tr_S, and define θ_k inductively. If no variables was introduced, $\theta_0 = id$. By induction, suppose θ_k defined up to the k first variables in tr_S such that $y\theta_k\phi_S\downarrow$ is a message if y is one of these k first variables. Let us prove that $R_x\phi_S\downarrow$ can be chosen so that $R_x\phi_S\downarrow$ is a message. Without loss of generality, as keys are atomic, we can choose a recipe R of $x\lambda_P$ such that $R = C[R_1, \ldots, R_m]$, C is a constructor-only context and for every $i \in \{1, \ldots, m\}$, R_i is destructor-only. As both $\downarrow$-reduction and $\Downarrow$-reduction contain only rules with destructors on top, we get that $\mathcal{T}^*_\phi(R) = C[\mathcal{T}^*_\phi(R_1), \ldots, \mathcal{T}^*_\phi(R_m)]$. As $R\phi\downarrow$ and C is constructor-only, for any i, $R_i\phi\downarrow$ is a message. By Lemmas 9 and 12 and Corollary 2, if $\mathcal{T}^*_\phi(R_i) = (R_i^*, \mathcal{R}_i)$, $R_i\phi\downarrow = (R_i^*\theta_k)\phi\downarrow$ and $R_i^*\phi_S\downarrow$ is a message. By induction hypothesis, we deduce $(R_i^*\theta_k)\phi_S\downarrow$ is a message. Taking $R_x = C[R_1^*\theta_k, \ldots, R_m^*\theta_k]$ ensures the result, and $\theta_{k+1} = \theta_k \cup \{x \mapsto R_x\}$. Then, suppose x appears in a key position inside $\mathsf{tr}_S\phi_S\downarrow$ but $x\theta\phi_S\downarrow$ is not an atom. As $\phi = \phi_S\lambda_P$, we would get that $x\lambda_P$ would appear in key position inside $\mathsf{tr}\phi\downarrow$ and $x\lambda_P$ would not be atomic, absurd as $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ and keys are atomic. Finally: $(\mathsf{tr}_S\theta)\phi\downarrow = (\mathsf{tr}_S\phi)\downarrow\lambda_P\downarrow$, as $\mathsf{tr}_S\phi = \mathsf{tr}_1$, $(\mathsf{tr}_S\theta)\phi\downarrow = \mathsf{tr}_1\lambda_P = \mathsf{tr}_1\tau = \mathsf{tr}_0\sigma$ and finally $(\mathsf{tr}_S\theta)\phi\downarrow = \mathsf{tr}\phi\downarrow$. □

Lemma 19 aimed at linking any concrete second-order trace of P to the valid second-order trace generated by the algorithm. The next ones, Lemmas 20 and 21 ensure similar properties for (tr_S, ψ_S) when it exists.

Lemma 20 (instances of valid symbolic traces) *If* (tr_S, ψ_S) *is a symbolic second-order trace of* Q*, then for every valid second-order substitution* θ*,* i.e. *such that for every* $x \mapsto R_x \in \theta$*,* R_x *is built from the initial knowledge of the attacker and the outputs which preceded the introduction of* x *in* tr_S*,* $R_x\psi_S\downarrow$ *is a message and for every variable* x *occurring in key position in* $\mathsf{tr}_S\psi_S\downarrow$*,* $R_x\psi_S\downarrow$ *is an atom, then there exists* ψ' *such that* $(\mathsf{tr}_S\theta, \psi') \in \mathsf{trace}(Q)$ *and* $\mathsf{tr}_S\theta\psi'\downarrow = \mathsf{tr}_S\psi_S\downarrow(\theta\psi')\downarrow$ *and* $\psi' = \psi_S(\theta\psi')\downarrow$*.*

Proof. Let us proceed by induction on n the length of tr_S^n and prove the following statements: $(\mathsf{tr}_S^n\theta, \psi^n) \in \mathsf{trace}(Q)$, $\mathsf{tr}_S^n\theta\psi^n\downarrow = \mathsf{tr}_S^n\psi_S^n\downarrow(\theta\psi^n)\downarrow$, $\psi^n = \psi_S^n(\theta\psi^n)\downarrow$ (which is a particular case of the previous item) and $\sigma^n = \sigma_S^n(\theta\psi^n)\downarrow$; where

$$\sigma^n = mgu((R_1\theta\psi^{n-1}\downarrow, u_1\rho_1), \ldots, (R_m\theta\psi^{n-1}\downarrow, u_m\rho_m))$$
$$\sigma_S^n = mgu((R_1\psi_S^{n-1}\downarrow, u_1\rho_1) \ldots, (R_m\psi_S^{n-1}\downarrow, u_m\rho_m))$$

and:

- tr^n denotes the truncation of tr at length n, m corresponds to the number of inputs in tr^n and ψ^n is the adequate subframe of ψ.
- the u_i are the input patterns of Q filtering the inputs of tr_S,
- ρ_i is the substitution applied to the remaining process after an input (θ in the description of our semantics rules). In particular, $\rho_1 = id$ and

$$\rho_{k+1} = mgu((R_1\theta\psi^{n-1}\downarrow, u_1\rho_1), \ldots, (R_k\theta\psi^{n-1}\downarrow, u_k\rho_k))$$

We can first note that

$$\begin{aligned}\sigma^n &= mgu((R_1\theta\psi^{n-1}\downarrow, u_1\rho_1), \ldots, (R_m\theta\psi^{n-1}\downarrow, u_m\rho_m))\\ &= mgu((R_1\theta\psi^{n-1}\downarrow, u_1), \ldots, (R_m\theta\psi^{n-1}\downarrow, u_m))\end{aligned}$$

by defining of the ρ_i and the unification algorithms for sets of pairs. Similarly,

$$\begin{aligned}\sigma_S^n &= mgu((R_1\psi_S^{n-1}\downarrow, u_1\rho_1) \ldots, (R_m\psi_S^{n-1}\downarrow, u_m\rho_m))\\ &= mgu((R_1\psi_S^{n-1}\downarrow, u_1) \ldots, (R_m\psi_S^{n-1}\downarrow, u_m))\end{aligned}$$

The induction itself:

- n=0: We consider only the initial knowledge of the attacker. As initial frames contain no variables, $\psi^0 = \psi_S^0$; because no input has been made, $\sigma^0 = \sigma_S^0 = \emptyset$; $\mathsf{tr}_S^0\theta\psi^0\downarrow = \mathsf{tr}_S^0\psi_S^0\downarrow(\theta\psi^0)\downarrow$; and finally, $(\mathsf{tr}^0\theta, \psi^0) \in \mathsf{trace}(Q)$ as it has length zero.
- Suppose we get the result up to some n: we make a disjunction on the $n+1$-th action in tr_S.
  - If tr_S^{n+1} ends by an output w (stored in ψ_S^{n+1}) on some channel c. As tr_S is a valid symbolic trace, any variable occurring in $\mathsf{w}\psi_S^n$ was first introduced in former inputs. $(\mathsf{tr}_S^n\theta, \psi^n) \in \mathsf{trace}(Q)$ by induction hypothesis; $\sigma^{n+1} = \sigma^n$ and $\sigma_S^{n+1} = \sigma_S^n$ as there is no new input; and finally, according to the input rule in our semantics, if v_{n+1} is the output pattern of the protocol specification being instantiated, $\mathsf{w}\psi_S^{n+1} = v_{n+1}\sigma_S^n$ and $\mathsf{w}\psi^{n+1} = v_{n+1}\sigma^n$. Hence $\mathsf{w}\psi_S^{n+1}(\theta\psi^n\downarrow) = \mathsf{w}\psi^{n+1}$, and $\psi^{n+1} = \psi_S^{n+1}(\theta\psi^n)\downarrow = \psi_S^{n+1}(\theta\psi^{n+1})\downarrow$ by induction hypothesis (and as the output does not introduce new variables in ψ^{n+1}). And thus $\mathsf{tr}_S^{n+1}\theta\psi^{n+1}\downarrow = \mathsf{tr}_S^{n+1}\psi_S^{n+1}\downarrow(\theta\psi^{n+1})\downarrow$, as the new action of the trace is the output for which the equality has been proved with the new frames ψ_S^{n+1} and ψ^{n+1}.

– If tr_S^{n+1} ends by an output of a channel c: c does not contain any variable and its value only depends on the number of channel outputted so far. As $\mathsf{tr}_S^n\theta\psi^n\downarrow = \mathsf{tr}_S^n\psi_S^n\downarrow(\theta\psi^n)\downarrow$ by induction hypothesis, no substitution is computed in the semantics nor any element added to the frame, we directly derive $\mathsf{tr}_S^{n+1}\theta\psi^{n+1}\downarrow = \mathsf{tr}_S^{n+1}\psi_S^{n+1}\downarrow(\theta\psi^{n+1})\downarrow$, $\psi^{n+1} = \psi^n$, $\sigma^{n+1} = \sigma^n$ (and similarly with their symbolic counterparts).

– If tr_S^{n+1} ends by an input with recipe R_{n+1} on some channel c. According to the input rule semantics, let u_{m+1} be the pattern to be match against R_{m+1} (in tr_S^{n+1}) and against $R_{m+1}\theta$ (in $\mathsf{tr}_S^{n+1}\theta$). Let σ_S^{n+1} (resp. σ^{n+1}) be the substitution introduced by the rule for tr_S^{n+1} (resp. $\mathsf{tr}_S^{n+1}\theta$). We have that:

$$\begin{aligned}\sigma_S^{n+1} = mgu((R_1\psi_S^n\downarrow, u_1)\ldots,(R_m\psi_S^n\downarrow, u_m),\\ (R_{m+1}\psi_S^n\downarrow, u_{m+1}))\\ \sigma^{n+1} = mgu((R_1\theta\psi^n\downarrow, u_1)\ldots,(R_m\theta\psi^n\downarrow, u_m),\\ (R_{m+1}\theta\psi^n\downarrow, u_{m+1}))\end{aligned}$$

These equalities can be rewritten as:

$$\begin{aligned}\sigma_S^{n+1} = mgu(mgu((R_1\psi_S^n\downarrow, u_1)\ldots,\\ (R_m\psi_S^n\downarrow, u_m)), (R_{m+1}\psi_S^n\downarrow, u_{m+1}))\\ \sigma^{n+1} = mgu(mgu((R_1\theta\psi^n\downarrow, u_1)\ldots,\\ (R_m\theta\psi^n\downarrow, u_m)), (R_{m+1}\theta\psi^n\downarrow, u_{m+1}))\end{aligned}$$

Note that ψ^n is actually equal to ψ^{n-1} on their common domain. Hence we get:

$$\begin{aligned}\sigma_S^{n+1} = mgu(\sigma_S^n, mgu(R_{m+1}\psi_S^n\downarrow, u_{m+1}))\\ \sigma^{n+1} = mgu(\sigma^n, mgu(R_{m+1}\theta\psi^n\downarrow, u_{m+1}))\end{aligned}$$

We now need to show that:

$$mgu(R_{m+1}\theta\psi^n\downarrow, u_{m+1}) = mgu(R_{m+1}\psi_S^n\downarrow, u_{m+1})(\theta\psi^n\downarrow)$$

Indeed, $R_{m+1}\theta\psi^n\downarrow = (R_{m+1}\psi^n)\downarrow(\theta\psi^n)\downarrow$ and as $dom(u_{m+1})\cap dom(\theta) = \emptyset$:

$$\begin{aligned}& mgu((R_{m+1}\psi_S^n)\downarrow(\theta\psi^n)\downarrow, u_{m+1})\\ = \;& mgu(R_{m+1}\psi_S^n\downarrow, u_{m+1})(\theta\psi^n\downarrow)\end{aligned}$$

Hence, as $\sigma^n = \sigma_S^n(\theta\psi^n)\downarrow$, we end up with $\sigma^{n+1} = \sigma_S^{n+1}(\theta\psi^n)\downarrow$, and because the last action of tr_S is an input, $\psi^n = \psi^{n+1}$, we get that $\sigma^{n+1} = \sigma_S^{n+1}(\theta\psi^{n+1})\downarrow$, as intended. Thus $\mathsf{tr}_S^{n+1}\theta\psi^{n+1}\downarrow = \mathsf{tr}_S^{n+1}\psi_S^{n+1}\downarrow(\theta\psi^{n+1})\downarrow$. In particular, $mgu(R_{m+1}\theta\psi^n\downarrow, u_{m+1}) \neq \bot$, so $(\mathsf{tr}_S^{n+1}\theta, \psi^{n+1}) \in \mathsf{trace}(Q)$ and $\psi^{n+1} = \psi_S^{n+1}(\theta\psi^{n+1})\downarrow$.

Thus our induction is complete, and for n equal to the length of tr_S, naming $\psi' = \psi^n$, we prove the desired result. □

Let $\sqsubseteq_k$ denotes the trace inclusion up to length k *i.e.*: $P \sqsubseteq_k Q$ if for any trace $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ of length (in term of number of actions) lesser than or equal to k, there exits $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$ such that $\phi \sim' \psi$.

Lemma 21 (from ϕ to ψ) *Let $n \in \mathbb{N}^*$. If $P \sqsubseteq_{n-1} Q$, given $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ of length $k \leq n$, $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$, tr_S and θ as defined in Lemma 19 and ψ_S such that (tr_S, ψ_S) is a symbolic second-order trace, then $\mathsf{tr}\psi\downarrow = \mathsf{tr}_S\psi_S\lambda_Q\downarrow$ and $\psi = \psi_S\lambda_Q$ where $\lambda_Q = (\theta\psi)\downarrow$.*

Proof. By definition of θ, for every recipe R of tr and its corresponding recipe R_S in tr_S, $R\phi\downarrow = R_S\theta\phi\downarrow$. As $P \sqsubseteq_{n-1} Q$, it implies $R\psi\downarrow = R_S\theta\psi\downarrow$ (if the last action is an output of a term, the variables are identical, if the last action is an input, their recipes are built on recipes from the frame generated from tr minus its last action, if the last action is the output of a channel, the channels are identical). By induction on tr and tr_S, we show both traces can share the same execution, thus leading to $\mathsf{tr}_S\theta\psi\downarrow = \mathsf{tr}\psi\downarrow$. From Lemma 20, as there exists ψ' such that $(\mathsf{tr}_S\theta, \psi')$ is also a second-order trace of Q and by induction on tr and tr_S, we show both traces can share the same execution, thus leading to $\psi = \psi'$; and setting $\lambda_Q = \theta\psi\downarrow$, we get that $\psi = \psi_S\lambda_Q$. Hence we finally get that $\mathsf{tr}\psi\downarrow = \mathsf{tr}_S\psi_S\lambda_Q\downarrow$. □

We are now able to prove Proposition 4 as stated in Section A.3.

Proposition 4 (completeness) *Let P and Q be two bounded protocols such that $P \not\approx Q$. The algorithm $\mathcal{A}_\mathcal{B}$ applied on P and Q returns a minimal (in term of number of actions) witness* tr *of non-equivalence.*

Proof. The proof is by induction on the length of traces. We suppose trace equivalence has been proved up to some length $n-1$ and consider a witness $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ of $P \not\approx_t Q$ of length n. Then we consider for instance the case where $P \not\sqsubseteq_t Q$, *i.e.* (tr, ψ) cannot belong in $\mathsf{trace}(Q)$ and the case where $\phi \not\sim' \psi$. The other cases are handled symmetrically.

From Lemma 19, we get the existence of (tr_S, ϕ_S) a symbolic second-order trace of P along with their substitutions λ_P and θ.

In the first case, suppose that $(\mathsf{tr}, \psi) \notin \mathsf{trace}(Q)$. If for every frame ψ_S, (tr_S, ψ_S) is not a symbolic trace of Q we have a witness. Otherwise let us assume that there exists ψ_S such that (tr_S, ψ_S) is a valid second-order symbolic trace of Q. For every $x \in dom(\theta)$, $x\theta\phi_S\downarrow$ is a message implies that $x\theta\psi_S\downarrow$ is a message, as equivalence between P and Q has been proved up to length $n-1$ (and $x\theta$ must be a recipe built from the frame after tr_S^{-1}). Now, if θ does not satisfy the property that for every variable x occurring in key position in $\mathsf{tr}_S\psi_S\downarrow$, $x\theta\psi_S\downarrow$ is an atom and thus there would exist a variable x such that x appears in key position in $\mathsf{tr}_S\psi_S\downarrow$ while $x\theta\psi_S\downarrow$ is not an atom. Equivalence up to length $n-1$ ensures $x\theta\phi_S\downarrow$ is not an atom. Thus, x should not occur in a key position in $\mathsf{tr}_S\phi_S\downarrow$, meaning $x \in K_{\psi_S} \setminus K_{\phi_S}$. Step 5 in the algorithm would then ensures that if $\lambda = \{^{\langle\omega,\omega\rangle}/_x\}$, $\mathsf{tr}_S\lambda$ is a witness of $P \not\sqsubseteq Q$, as $(\mathsf{tr}_S\lambda, \phi_S\lambda)$ is valid symbolic trace of P but not of Q, as $\langle\omega,\omega\rangle$ would appear in key position in $(\mathsf{tr}_S\lambda)(\psi_S\lambda)\downarrow$. In the following we will then assume θ satisfy this property. Using Lemma 20, we get that $(\mathsf{tr}_S\theta, \psi') \in \mathsf{trace}(Q)$ and $\mathsf{tr}_S\theta\psi'\downarrow = \mathsf{tr}_S\psi_S\downarrow(\theta\psi'\downarrow)$. Because $P \sqsubseteq_{n-1} Q$, there exists ψ'' such that $(\mathsf{tr}^{-1}, \psi'') \in \mathsf{trace}(Q)$ and $\mathsf{tr}_S^{-1}\theta\psi''\downarrow = \mathsf{tr}^{-1}\psi''\downarrow$. Indeed, by definition of θ, for every recipe R of tr and its corresponding recipe R_S in tr_S, $R\phi\downarrow = R_S\theta\phi\downarrow$. As $P \sqsubseteq_{n-1} Q$, it implies $R\psi''\downarrow = R_S\theta\psi''\downarrow$ (if the last action is an output of a term, the variables are identical, if the last action is an input, their recipes are built on recipes from the frame generated from tr minus its last action, if the last action is the output of a channel, the channels are identical). By induction on tr^{-1} and tr_S^{-1}, we show both traces can share the same execution, thus leading to $\mathsf{tr}_S^{-1}\theta\psi''\downarrow = \mathsf{tr}^{-1}\psi''\downarrow$. Moreover after executing $\mathsf{tr}_S^{-1}\theta$ and tr^{-1}, Q ends up in the same configuration (by following the same execution). Then, if tr's last action is an output of a term, because $(\mathsf{tr}_S\theta, \psi') \in \mathsf{trace}(Q)$, there exists ψ such that $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$, contradiction. If its last action is an input on some channel c with a recipe R, as we already established that $R_S\theta\psi''\downarrow = R\psi''\downarrow$ and their configurations are identical, there exists ψ such that $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$, also reaching a contradiction. If

the last action is an output of channel, $(\mathsf{tr}_S, \theta') \in \mathsf{trace}(Q)$ directly implies $(\mathsf{tr}, \psi'') \in \mathsf{trace}(Q)$, which is also a contradiction. Hence (tr_S, ψ_S) is not a valid second-order trace of Q if $(\mathsf{tr}, \psi) \notin \mathsf{trace}(Q)$ for any frame ψ.

Assume $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$ for some ψ and $\phi \not\sim' \psi$ for any such ψ. In the second case, we need to prove either that if there is a recipe leading to a message in P but not in Q the algorithm will yield an attack; or that if an equality holds in P but not in Q, the algorithm can derive two equalities which will hold in P and not in Q; or that if a recipe leads to an atom in ϕ but not in ψ, the algorithm witnesses it. As $(\mathsf{tr}_S, \psi_S) \in \mathsf{trace}(Q)$ in this case, from Lemma 21 and because Q is determinate, we can derive the existence of λ_Q such that $\psi = \psi_S\lambda_Q$, which is a required hypothesis of Lemma 15.

In the first subcase, let R be a ϕ-precompact recipe such that $R\phi\downarrow$ is a message but $R\psi\downarrow$ is not. Let us then define $(R^*, \mathcal{R}_1) = \mathcal{T}^*_\phi(R)$ and $(R'^*, \mathcal{R}_2) = \mathcal{T}^*_\psi(R)$. We operate the following disjunction:

1. $\psi_S \not\models \mathcal{R}_1$ or $\phi_S \not\models \mathcal{R}_2$. By Lemma 6, $\phi_S \models \mathcal{R}_1$. Consider for example an equality in $\mathcal{R}_1$ witnessing $\psi_S \not\models \mathcal{R}_1$: this equality is captured by the algorithm at step 6, hence leading to a witness of non-equivalence. The other case is treated symmetrically.

2. else, if $\psi_S \models \mathcal{R}_1$ and $\phi_S \models \mathcal{R}_2$: we can apply Lemma 14, thus $R^* = R'^*$. If $R^*\psi_S\downarrow$ is not a message, as R^* is ϕ_S-compact (because R is ϕ-precompact and of Lemma 13), we have a witness of non-inclusion provided by the algorithm. So let us assume that $R^*\psi_S\downarrow$ is a message. We can now apply Lemma 17. With the same notations, either there exists a ϕ_S-compact recipe R_0 such that $R_0\phi_S\downarrow$ is a message while $R_0\psi_S\downarrow$ is not, in which case we directly get a witness of non-inclusion; or we can consider the next case.

We now deal with the subcase where an equality holds in ϕ but not in ψ. Let R_1 and R_2 be two ϕ-precompact recipes such that $R_1\phi\downarrow = R_2\phi\downarrow$ and $R_1\psi\downarrow \neq R_2\psi\downarrow$. Let us define $(R_1^*, \mathcal{R}_1) = \mathcal{T}^*_\phi(R_1)$, $(R_2^*, \mathcal{R}_2) = \mathcal{T}^*_\phi(R_2)$, $(R_1'^*, \mathcal{R}'_1) = \mathcal{T}^*_\psi(R_1)$ and $(R_2'^*, \mathcal{R}'_2) = \mathcal{T}^*_\psi(R_2)$. As previously, we can operate the following disjunction:

1. $\psi_S \not\models \mathcal{R}_1$ or $\psi_S \not\models \mathcal{R}_2$ or $\phi_S \not\models \mathcal{R}'_1$ or $\phi_S \not\models \mathcal{R}'_2$: see the previous subcase, we can directly create a witness of non-equivalence.

2. $\psi_S \models \mathcal{R}_1$, $\psi_S \models \mathcal{R}_2$, $\phi_S \models \mathcal{R}'_1$ and $\phi_S \models \mathcal{R}'_2$: we can now apply Lemma 14 twice and get that $R_1'^* = R_1^*$ and $R_2'^* = R_2^*$. From that we apply Lemma 16 and get that there exists σ mgu of two ϕ_S-compact recipes such that such that $R_1^*\phi_S\sigma\downarrow = R_2^*\phi_S\sigma\downarrow$. Thus we can choose $\sigma_1 = \sigma$ at step 2 in the algorithm. And if, for the sake of clarity, we name tr_E, ϕ_E and ψ_E the trace and frames (respectively) introduced at step 3 of the algorithm, it ensures that $R_1^*\phi_E\downarrow = R_2^*\phi_E\downarrow$. From Lemmas 19 and 21, we get new first-order substitutions λ'_P (for (tr_E, ϕ_E)) and λ'_Q (for (tr_E, ψ_E)). From there, we apply the same reasoning as before, as an equality still holds in ϕ but not in ψ, using $\mathcal{T}^*$ and making three separate cases. The first two are identical. In the third one, we now have that we can once again apply Lemma 14, followed by Lemma 16. In this case, $\sigma = id$ (by construction of ϕ_E), and thus we get an equality $R_1^* = R_2^*$ of compact recipes in ϕ_E and an inequality in ψ_E (Lemma 15).

We finally deal with the case of testing atomicity of a term: suppose there exists R a ϕ-precompact recipe such that $R\phi\downarrow$ is an atom while $R\psi\downarrow$ is not. Let $(R^*, \mathcal{R}_1) = \mathcal{T}^*_\phi(R)$ and $(R'^*, \mathcal{R}_2) = \mathcal{T}^*_\psi(R)$.

- if $\phi_S \not\models \mathcal{R}_2$ or $\psi_S \not\models \mathcal{R}_1$: as before, we get a witness of non-equivalence in our algorithm,

- else, if $\phi_S \models \mathcal{R}_2$ and $\psi_S \models \mathcal{R}_1$, by Lemma 14, $R^* = R'^*$. As R^* is ϕ_S-compact (Lemma 13), $R\phi_S\downarrow$ is not a variable, and in particular $R^*\phi_S\downarrow = R\phi\downarrow$ (by Lemma 9). Hence $R^*\phi_S\downarrow$ is an atom. Now, if $R^*\psi_S\downarrow$ is a variable, $R^* = x$ holds in ψ_S for some variable x but not in ϕ_S, yielding a witness of non static equivalence. Else, if $R^*\psi_S\downarrow$ is a composed term, we get a witness of non static inclusion as $R^*\psi_S\downarrow$ would not be an atom. Finally, if $R\psi_S\downarrow$ is an ground atom, as in that case $R^*\psi_S\downarrow = R\psi\downarrow$ (by Lemma 9), but $R\psi\downarrow$ is not an atom which cannot happen as it would contradict our hypothesis.

In conclusion, we found a witness of length n of $P \not\approx_t Q$ assuming trace equivalence up to length $n-1$: hence we the algorithm derived a shortest witness in terms of number of actions. □

C Proofs of Theorem 1 and Proposition 1

In Section B, we proved the completeness of the procedure described in Section A. To prove Theorem 3, and thus Proposition 1, we still need to prove this algorithm actually preserves types. Once done, we will be able to finally prove Theorem 1. Section **??** will focus on these results while Section E will provide the proof of Corollary 1 for tagged protocols.

A stronger version of Proposition 1 is actually proven with Theorem 3: not only there exists such an algorithm, but the algorithm described in Section A.2 does satisfy all the necessary conditions.

Theorem 3 *Let P and Q be two bounded protocols type-compliant w.r.t. $(\mathcal{T}_1, \delta_1)$ and $(\mathcal{T}_2, \delta_2)$ respectively, and such that $P \not\approx Q$. Assume the algorithm $\mathcal{A}_{\mathcal{B}}$ uses a type-preserving reachability blackbox $\mathcal{B}$ and a well-typed renaming ρ at step 3. Then $\mathcal{A}_{\mathcal{B}}(P, Q)$ returns a trace tr such that*

- *either $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ for some ϕ and (tr, ϕ) is pseudo-well-typed w.r.t. $(\mathcal{T}_1, \delta_1)$;*

- *or $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$ for some ψ and (tr, ψ) is pseudo-well-typed w.r.t. $(\mathcal{T}_2, \delta_2)$.*

Proof. Assume here the witness of non-equivalence is a trace of P. Because the reachability blackbox preserves types, the trace tr_1 at step 1 is well-typed. At step 2, unification can only happen on a pair (s, t) of ciphertexts with variables: $s, t \in ESt(\mathsf{tr}_1) \subseteq ESt(\mathsf{tr})\sigma_1$. Thus there exists two terms $s', t' \in ESt(P)$ such that $s = s'\sigma_1$ and $t = t'\sigma_1$. Since P is type-compliant w.r.t. $(\mathcal{T}_1, \delta_1)$, $\delta_1(s') = \delta_1(t')$. By definition of a typing system, σ_1 is well-typed and then $\delta_1(t) = \delta_1(s)$ and thus their unifier is well-typed. Hence $\mathsf{tr}_1\sigma_1$ is well-typed, and applying the reachability blackbox and a well-typed renaming leads to (tr, ϕ) being well-typed. If $\mathcal{A}_{\mathcal{B}}(P, Q)$ outputs a witness tr at step 5 and c_0 is the constant replaced by $\langle\omega, \omega\rangle$, as (tr, ϕ) is well-typed and thus $(\mathsf{tr}\lambda, \phi\lambda) \in \mathsf{trace}(P)$, where $\lambda = \{^{\langle\omega,\omega\rangle}/_{\mathsf{c}_0}\}$ is pseudo-well-typed. The symmetric case with traces of Q is handled similarly, as Q is type-compliant w.r.t. $(\mathcal{T}_2, \delta_2)$. □

The proof of Proposition 1 is then a direct consequence of Theorem 3.

Theorem 1 *Let P and Q be two determinate protocols type-compliant w.r.t. $(\mathcal{T}_1, \delta_1)$ and $(\mathcal{T}_2, \delta_2)$ respectively. We have that $P \not\approx Q$ if, and only if, there exists a witness of non-equivalence* tr *such that:*

- *either* $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ *for some* ϕ *and* (tr, ϕ) *is pseudo-well-typed w.r.t.* $(\mathcal{T}_1, \delta_1)$*;*
- *or* $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$ *for some* ψ *and* (tr, ψ) *is pseudo-well-typed w.r.t.* $(\mathcal{T}_2, \delta_2)$.

Proof. If P or Q contain replication and $P \not\approx_t Q$, there exists a minimal witness tr of length n of this non-equivalence. Consider P' (resp. Q') the unfolding of P where every occurrence of a subprocess $!R$ is replaced by $R|\ldots|R$ ($n+1$ times), with α-renaming to avoid name and variable capture. Because tr is of length n, $\mathcal{A}(P', Q')$ or $\mathcal{A}(Q', P')$ would yield a witness of $P' \not\approx_t Q'$ of length lesser than n, as Proposition 4 ensures the algorithm returns the shortest witness of non-equivalence. Then, as P' and Q' were unfolded $n+1$ times, this witness is also a witness of $P \not\approx_t Q$. We now need to prove that P' is type-compliant w.r.t. $(\mathcal{T}_1, \delta_1)$ and Q' is type-compliant w.r.t. $(\mathcal{T}_2, \delta_2)$. For instance, let $s', t' \in ESt(P')$ and σ' such that $t'\sigma' = s'\sigma'$. Because P' can be α-renamed so as to use common variable and names with $\mathsf{unfold}^2(P)$, there exist $s, t \in ESt(\mathsf{unfold}^2(P))$, renamings of s' and t' respectively, and σ, renaming of σ', such that $s\sigma = t\sigma$, $\delta_1(s) = \delta_1(s')$ and $\delta_1(t) = \delta_1(t')$ (as α-renaming preserves types). By definition of type-compliant we can conclude that $\delta_1(t) = \delta_1(s)$ and thus P' is type-compliant w.r.t. $(\mathcal{T}_1, \delta_1)$. The same reasoning also applies to Q' w.r.t. $(\mathcal{T}_2, \delta_2)$. Hence P' and Q' are both type-compliant w.r.t. to their respective typing systems and without replication. Hence we only need to deal with the case where P and Q do not use replication. Theorem 3 and Proposition 4 ensure that $P \not\approx_t Q$ if, and only if, $\mathcal{A}(P, Q)$ yields a witness, which is well-typed for at least one of these protocols according to Theorem 3. □

D Proof of Theorem 2

In this section, we prove the Lemmas introduced in Section 4 and ultimately Theorem 2 itself.

Lemma 2 *A simple protocol is determinate.*

Proof. We prove by induction that if P is a simple protocol, for any sequence tr of actions, there exists at most one configuration $(\mathcal{P}; \phi)$ up to τ transitions such that $(P; \emptyset) \stackrel{\mathsf{tr}}{\Longrightarrow} (\mathcal{P}, \phi)$, *i.e.* if $(P; \emptyset) \stackrel{\mathsf{tr}}{\Longrightarrow} (\mathcal{P}_1, \phi_1)$ and $(P; \emptyset) \stackrel{\mathsf{tr}}{\Longrightarrow} (\mathcal{P}_2, \phi_2)$ then there exists configuration $(\mathcal{P}_3, \phi_1)$ such that $(\mathcal{P}_1, \phi_1) \stackrel{\tau}{\rightarrow}{}^* (\mathcal{P}_3, \phi_1)$ and $(\mathcal{P}_2, \phi_2) \stackrel{\tau}{\rightarrow}{}^* (\mathcal{P}_3, \phi_1)$. Thus, *a fortiori*, P is determinate. Let us proceed with the induction:

- if tr is empty, $(\mathcal{P}, \phi) = (P, \emptyset)$,
- suppose there exists a unique configuration up to τ transitions $(\mathcal{P}', \phi')$ such that $(P, \emptyset) \stackrel{\mathsf{tr}'}{\Longrightarrow} (\mathcal{P}', \phi')$ (if $(\mathcal{P}', \phi')$ does not exits, the results holds directly). If $\mathsf{tr} = \mathsf{tr}'.\alpha$ for some action α and there exists $(\mathcal{P}, \phi)$ such that $(\mathcal{P}', \phi') \stackrel{\alpha}{\Longrightarrow} (\mathcal{P}, \phi)$, let us show that $(\mathcal{P}, \phi)$ is unique up to τ transitions.
  - if $\alpha = \mathsf{out}(c_j, ch_i)$: as P is simple, the only process able to fire that transition is $\mathsf{new}\ c.\mathsf{out}(c_j, c).B_j$ for some c. Hence $(\mathcal{P}, \phi) = (\mathcal{P}' \cup$

$\{B_j[c/ch_i]\} \setminus \{\mathsf{new}\ c.\mathsf{out}(c_j, c).B_j\})$ up to new occurrences of $\mathsf{new}\ c.\mathsf{out}(c_k, c).B_k$ achieved by τ transitions (replication rule).

- if $\alpha = \mathsf{out}(ch_i, \mathsf{w}_j)$: because B_i is a sequence of actions and ch_i is unique for every use of the output channel rule, the number of occurrences of ch_i in tr' pinpoints to exactly one output in B_i that can be fired. Because our semantics is deterministic ($\mathsf{w}\phi = t$ depends only on $\mathcal{P}'$), up to replication of branches of P, $(\mathcal{P}, \phi' \cup \{\mathsf{w} \triangleright t\})$ is unique.
- if $\alpha = \mathsf{in}(ch_i, R)$): once again, B_i is a sequence of actions and ch_i is unique for every use of the output channel rule, the number of occurrences of ch_i in tr' pinpoints to exactly one input $\mathsf{in}(ch_i, u)$ in B_i that can be fired. As in that case $\phi = \phi'$, by unicity of ϕ', and as our semantics is deterministic (there exists only one substitution θ such that $R\phi'\downarrow = u\theta$), up to replication of branches of P, $(\mathcal{P}, \phi')$ is unique.

□

Proposition 5 *There exists a set of constants and name with finitely many elements of any type such that if P and Q are well-formed and simple and $P \not\approx_t Q$ then there exists a pseudo-well-typed witness* tr *of non-equivalence which uses only those constants and names.*

Proof. By Lemma 2 and Theorem 1, there exists a well-typed witness $\bar{\mathsf{tr}}_1$ of non-equivalence.

In the following, we assume that $\bar{\mathsf{tr}}_1$ has been discovered when applying the equivalence algorithm $\mathcal{A}_{\mathcal{B}}(P, Q)$. The symmetric case is handled in the same fashion. Moreover, we can also assume the blackbox $\mathcal{B}$ used does not introduce new variables or constants as the procedure described [16], so that we can also assume that any constant in $\bar{\mathsf{tr}}_1\phi_0\downarrow$ (where $(\bar{\mathsf{tr}}_1, \phi_0) \in \mathsf{trace}(P)$) is either an element of $\Sigma_0^P \cup \mathcal{N}^P$ (the constants and names in P) or a constant introduced at step 3 of our algorithm. Let $A = \{\alpha_1, \ldots, \alpha_n\}$ be the set of such constants. We also consider a set of special constant $\mathcal{C} = \bigcup_{\tau \in \mathcal{T}^P} \mathcal{C}_\tau$ where $\mathcal{C}_\tau = \{c_1^\tau, c_2^\tau, c_3^\tau\}$ and $\delta_1(c_i^\tau) = \tau$. Note that $\mathcal{T}_0(\Sigma_{\mathsf{c}}, \cup\Sigma_0^P \cup \mathcal{N}^P \cup \bigcup_\tau \mathcal{C}_\tau)$ is such that there are only finitely many terms of any given type.

Claim: there exists a (total) renaming ρ from A to $\bigcup_\tau \mathcal{C}_\tau$ such that $\bar{\mathsf{tr}}_1\rho$ is a well-typed witness of $P \not\approx Q$ and for any term t of $\bar{\mathsf{tr}}_1\rho\phi_0\downarrow$, $t \in \mathcal{T}_0(\Sigma_{\mathsf{c}}, \cup\Sigma_0^P \cup \mathcal{N}^P \cup \bigcup_\tau \mathcal{C}_\tau)$; where ϕ_0 is such that $(\bar{\mathsf{tr}}_1, \phi_0) \in \mathsf{trace}(P)$.

Let ρ_0 be the special renaming such that for any i, if $\delta_1(\alpha_i) = \tau_i$, $\alpha_i\rho_0 = c_1^{\tau_i}$.

$\bar{\mathsf{tr}}_1$ being a witness of $\bar{P} \not\approx \bar{Q}$, several cases can occur:

- there exists ϕ and ψ such that $(\bar{\mathsf{tr}}_1, \phi) \in \mathsf{trace}(P)$ and $(\bar{\mathsf{tr}}_1, \psi) \in \mathsf{trace}(Q)$; $\phi \not\sim \psi$ and, for instance, there exist two recipes R_1 and R_2 such that $R_1\phi\downarrow = R_2\phi\downarrow$ but $R_1\psi\downarrow \neq R_2\psi\downarrow$ and all of them are messages. Let us examine $R_1\psi\downarrow$ and $R_2\psi\downarrow$. If the two terms do not share the same constructors, then $R_1(\psi\rho_0)\downarrow \neq R_2(\psi\rho_0)\downarrow$, but $R_1(\phi\rho_0)\downarrow = R_2(\phi\rho_0)\downarrow$ (as the collapsing of variables only add equalities to the frame). Now, if the two terms share the same constructors, there must exist a leaf position p in them such that $R_1\psi\downarrow|_p \neq R_2\psi\downarrow|_p$. Let us call t and s these terms respectively. If s or t is *not* an α_i for some i, then $s\rho_0 \neq t\rho_0$ as the constants in $\bigcup_\tau \mathcal{C}_\tau$ are fresh. As in the previous case, we get that $R_1(\psi\rho_0)\downarrow \neq R_2(\psi\rho_0)\downarrow$, but $R_1(\phi\rho_0)\downarrow = R_2(\phi\rho_0)\downarrow$. Else, assume $s = \alpha_1$ and

$t = \alpha_2$, consider the renaming ρ such that, if for any i, $\delta_1(\alpha_i) = \tau_i$, $\alpha_1\rho = c_1^{\tau_1}$, $\alpha_2\rho = c_2^{\tau_2}$ and $\alpha_j\rho = c_3^{\tau_j}$ for any $j > 2$. Thus $s\rho \neq t\rho$ as the constants in $\bigcup_\tau \mathcal{C}_\tau$ are fresh, and finally we get that $R_1(\psi\rho)\downarrow \neq R_2(\psi\rho)\downarrow$, but $R_1(\phi\rho)\downarrow = R_2(\phi\rho)\downarrow$.

- Or there exists ϕ and ψ such that $(\bar{\mathsf{tr}}_1, \phi) \in \mathsf{trace}(P)$ and $(\bar{\mathsf{tr}}_1, \psi) \in \mathsf{trace}(Q)$; $\phi \not\sim \psi$ and there exists a minimal (in term of size) recipe R such that, for instance, $R\phi\downarrow$ is message while $R\psi\downarrow$ is not. If $R\psi\downarrow$ contains a element $\mathsf{dec}(\langle s, t\rangle)$ or $\mathsf{proj}_i(\mathsf{enc}(s, t))$, then $R(\psi\rho_0)\downarrow$ is not a message either, while $R(\phi\rho_0)\downarrow$ still is. Else, $R\psi\downarrow = \mathsf{dec}(\mathsf{enc}(u, v), w)$ for some terms u, v and w (by minimality) with $v \neq w$: as keys are atomic, v and w are atoms. As in the case with equalities, we can define ρ such that $v\rho \neq w\rho$, and thus $R(\psi\rho)\downarrow$ is not a message, while $R(\phi\rho)\downarrow$ is (as ρ only introduces new equalities).

- Or, finally, there exists ϕ such that $(\bar{\mathsf{tr}}_1, \phi) \in \mathsf{trace}(P)$ but for every ψ, $(\bar{\mathsf{tr}}_1, \psi) \notin \mathsf{trace}(Q)$ (the symmetric case is handled identically). If $\bar{\mathsf{tr}}_1$ end with an output, the renaming ρ_0 is adequate. So let us assume that the last action of $\bar{\mathsf{tr}}_1$ is $\mathsf{in}(c, R)$. Because protocols are simple, there exists at most one term u_P in the execution of $\bar{\mathsf{tr}}_1$ in P, and at most one term u_Q in the same execution in Q such that there exists $\exists\theta(R\phi\downarrow = u_P\theta)$, but $\forall\theta'(R\psi\downarrow \neq u_Q\theta')$. As u_Q may contain several occurrences of variables, we need to be careful to define a renaming ρ. If there exists a position p in u_Q which is not a leaf such that $u_Q|_p \neq R\psi\downarrow|_p$, then $R(\psi\rho_0)\downarrow$ and $u_Q\rho_0$ are not unifiable (they disagree on already present constructors). Else if there exists a position p in u_Q which is a leaf but not a variable such that $u_Q|_p \neq R\psi\downarrow|_p$: we define ρ as in the first subcase when dealing with an inequality without variable. Finally, if $mgu(R(\psi\rho_0)\downarrow, u_Q) = \bot$ we are done and can just take $\rho = \rho_0$; else, *i.e.* $mgu(R\psi\downarrow, u_Q) = \bot$ and $mgu(R(\psi\rho_0)\downarrow, u_Q) \neq \bot$, there exist two leaves with positions p_1 and p_2 in $R\psi\downarrow$ which corresponds to positions below variables in u_Q such that $R\psi\downarrow|_{p_1} \neq R\psi\downarrow|_{p_2}$ but $R(\psi\rho_0)\downarrow|_{p_1} = R(\psi\rho_0)\downarrow|_{p_2}$: thus we can assume $R\psi\downarrow|_{p_1} = \alpha_1$ and $R\psi\downarrow|_{p_2} = \alpha_2$. We can now define ρ such that, if for any i, $\delta_1(\alpha_i) = \tau_i$, $\alpha_1\rho = c_1^{\tau_1}$, $\alpha_2\rho = c_2^{\tau_2}$ and $\alpha_j\rho = c_3^{\tau_j}$ for any $j > 2$. Then $R(\psi\rho)\downarrow|_{p_1} \neq R(\psi\rho)\downarrow|_{p_2}$ and $mgu(R(\psi\rho)\downarrow, u_Q) = \bot$, while $mgu(R(\phi\rho)\downarrow, u_Q) \neq \bot$ as ρ only introduces new equalities.

In every case, $\bar{\mathsf{tr}}_1\rho$ is valid trace of $\bar{P}$, and $\bar{\mathsf{tr}}_1\rho$ is a trace of P with frame ϕ_0 which is well-typed (ρ is well-typed) and for any term t of $\bar{\mathsf{tr}}_1\rho\phi_0\downarrow$, $t \in \mathcal{T}_0(\Sigma_\mathsf{c}, \cup\Sigma_0^P \cup \mathcal{N}^P \cup \bigcup_\tau \mathcal{C}_\tau)$. □

Theorem 2 *The problem of deciding whether two simple protocols P and Q, type-compliant w.r.t. some finite typing systems $(\mathcal{T}_1, \delta_1)$ and $(\mathcal{T}_2, \delta_2)$ are trace equivalent* (i.e. $P \approx Q$) *is decidable.*

Proof. By Proposition 5, if $P \not\approx_t Q$ there exists a well-typed witness of this non-equivalence built on a special set of terms. Let for instance $(\mathsf{tr}^0, \phi^0) \in \mathsf{trace}(P)$ be a minimal such witness and define $\mathsf{tr} = \mathsf{tr}^0_{-1}$ and ϕ the frame such that $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$. The minimality of tr^0 implies there exists ψ such that $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$

and $\phi \sim_s \psi$.

$$\begin{array}{rl} P = & !\mathsf{new}\ c'_1.\mathsf{out}(c_1,c'_1).B_1 \mid \ldots \mid !\mathsf{new}\ c'_m.\mathsf{out}(c_m,c'_m).B_m \\ & \mid B_{m+1} \mid \ldots \mid B_{m+n} \\ Q = & !\mathsf{new}\ c'_1.\mathsf{out}(c_1,c'_1).B'_1 \mid \ldots \mid !\mathsf{new}\ c'_m.\mathsf{out}(c_m,c'_m).B'_m \\ & \mid B'_{m+1} \mid \ldots \mid B'_{m+n'} \end{array}$$

(We can safely assume the same number of restricted channels (m) in P and Q, as non-equivalence would trivially hold otherwise.) We define a function proj_P which takes a channel c_i and returns $\{\mathsf{tr}|_{c_i,c'_i}\}_{c'_i}$ the set of subtraces of tr corresponding of actions on channel c'_i and the action $\mathsf{new}\ c'_i.\mathsf{out}(c_i,c'_i)$ which originated it. It can be extended naturally to channels c_{m+1} to c_{m+n} (in that case, in absence of replication, each set is a singleton, which, for the sake of uniformity, we will denote by $\{\mathsf{tr}|_{c_i,c'_i}\}_{c'_i}$ for $i \in \{m+1,\ldots,m+n\}$ too). We can similarly define proj_Q. We now claim:
Claim 1: $(\mathsf{tr},\psi) \in \mathsf{trace}(Q) \Rightarrow \mathsf{proj}_P = \mathsf{proj}_Q$, as subprocesses B_i and B'_i use public fresh channels such that each c'_i is spawned from a unique channel c_i.

Moreover we define a relation $\mathcal{R}_P$ on elements of $img(\mathsf{proj}_P)$. We say that $\mathsf{tr}|_{c_i,c'_i} \preceq_P \mathsf{tr}|_{c_i,c'_i}$ if, and only if, $\mathsf{tr}|_{c_i,c'_i}\phi\downarrow$ is a prefix of $(\mathsf{tr}|_{c_i,c'_i}\phi\downarrow)\{^{c_i}/_{c'_i}\}$. $\mathcal{R}_P$ is then the symmetric closure of $\preceq_P$; and is an equivalence relation. Symmetrically, we can define a relation $\mathcal{R}_Q$ with ψ and make the following claim:
Claim 2: $\mathcal{R}_P = \mathcal{R}_Q = \mathcal{R}$, as tr^0 is a minimal witness of $P \not\approx Q$.

Given an action α we map it to $\beta(\alpha)$ its occurrence index in tr. Consequently we can define $\min(\alpha,\alpha')$ as the action with the lowest occurrence index; and lift it to sequences of actions and sets of sequences: $\min(\alpha_1,\ldots,\alpha_n,\alpha'_1,\ldots,\alpha'_n) = \min(\alpha_1,\alpha'_1).\ldots.\min(\alpha_n,\alpha'_n)$. Moreover, if ϵ denotes the empty sequence, $\min(A,\epsilon) = \min(\epsilon,A) = A$ for any sequence A, and we are thus able to define $\min$ for sequences of different lengths. We define quite differently the maximum of two sequences A and A' by $\max(A,A') = A.A' \setminus \min(A,A')$, *i.e.* the (ordered) sequence of actions of A and A' which are not minimal. We finally define a function merge taking as arguments any subtrace tr_0 of tr and tow sequences of actions A and A' such that $A\,\mathcal{R}\,A'$ and maps it to $\mathsf{merge}(\mathsf{tr}_0,A,A') = (\mathsf{tr}_0 \setminus \max(A,A'))\sigma_{\min}$ where $\sigma_{\min}$ is the substitution which maps any $\mathsf{w} \in dom(\phi) \cap (\mathsf{out}(A) \cup \mathsf{out}(A'))$ to the $\mathsf{w}' \in dom(\phi) \cap \mathsf{out}(\min(A,A'))$ at the same position *i.e* such that $\mathsf{out}(c,\mathsf{w}') = \min(\mathsf{out}(c,\mathsf{w}),\mathsf{out}(c,\mathsf{w}'))$, where $\mathsf{out}(A)$ denotes the set of variables in outputs of A. In particular, if $\mathsf{tr}' = \mathsf{merge}(\mathsf{tr}_0,A,A')$, then there exist ϕ' and ψ' such that $(\mathsf{tr}',\phi') \in \mathsf{trace}(P)$, $(\mathsf{tr}',\psi') \in \mathsf{trace}(Q)$, $\mathsf{tr}'\phi'\downarrow = (\mathsf{tr}_0 \setminus \max(A,A'))\phi\downarrow$ and $\mathsf{tr}'\psi'\downarrow = (\mathsf{tr}_0 \setminus \max(A,A'))\psi\downarrow$.

Let $[\mathsf{tr}_1],\ldots,[\mathsf{tr}_M]$ be the equivalence classes of $\mathcal{R}$, then, by minimality of tr^0, each class has at exactly one element. If some class $[\mathsf{tr}_i]$ were to have two elements A and A', $\mathsf{merge}(\mathsf{tr},A,A').(\alpha\sigma_{\min})$ would provide a shorter witness of $P \not\approx Q$, where α is the last action of tr^0 (which was excluded in tr) and $\sigma_{\min}$ renames its variable if needed, to ensure being a trace.

From Proposition 5, we know there exists a set of atoms such that there exists only finitely element of the same type and such that any pseudo-well-typed trace only uses those. As the type systems we consider are finite, there exists only finitely many terms of each type. And finally, as the trace is pseudo-well-typed, each type in $\mathsf{tr}^0\phi^0\downarrow$ appears in P. Hence there are only finitely types and finitely many terms that can occur in any pseudo-well-typed trace of P: let T be that number. Thus we claim that $M \leq (n+m)\times T^B$ where B is the maximal length (in terms of number of actions) of a parallel branch in P, *i.e.* $\max_i |\mathsf{tr}_i|$. Indeed a class is defined by its sequence of actions (bounded by the maximum number of actions in any branch of P) and the first-order

terms it contains (which is bounded by the total number of existing eligible terms). As there are n branches in P, of length at most M, there are at most $(n+m)T^B$ different first-order sequences in P (up to prefixes). Then, as $|\mathsf{tr}^+| \leq M \times B$ (tr^+ contains exactly one representative for each equivalence class, each being of length at most B), we finally get $|\mathsf{tr}| \leq (n+m) \times B \times T^B$. Hence, the maximal length of a well-typed witness of $P \not\approx_t Q$ is $N = (n+m)BT^B + 1$, and the number of such traces in $\mathsf{trace}(P)$ is bounded by $\sum_{i=0}^{N} T^i$, which provides a straightforward algorithm to decide trace equivalence. □

E Proof of Corollary 1

For every tagged protocol, we can define a induced typing system which is useful to prove Corollary 1.

Definition 19 *Let P be a tagged protocol, and σ_P the substitution witnessing this fact. Let Σ_P, $\mathcal{N}_P$, $\mathcal{X}_P$ be respectively the constants, names, and variables occurring in P. We consider the function δ_P, inductively defined on $\mathcal{T}(\Sigma_\mathsf{c}, \Sigma_P \cup \mathcal{N}_P \cup \mathcal{X}_P)$ as follows:*

- $\delta_P(x) = \overline{x\sigma_P}$ *for any variable that occurs in P;*
- $\delta_P(a) = a$ *for any name, constant that occurs in P.*
- $\delta(\mathsf{f}(t_1, t_2)) = \mathsf{f}(\delta(t_1), \delta(t_2))$ *for* $\mathsf{f} \in \{\mathsf{enc},\ \langle\ \rangle\}$.

The image of δ_P is a set of types, denoted $\mathcal{T}_p$. The function δ_P is then extended arbitrarily to the remaining names, variables, constants such that there is an infinite set of names, variables, constants of each type in $\mathcal{T}_p$. This extends δ_P on $\mathcal{T}(\Sigma_\mathsf{c}, \Sigma_0 \cup \mathcal{N} \cup \mathcal{X})$ using the recursive definition: $\delta(\mathsf{f}(t_1, t_2)) = \mathsf{f}(\delta(t_1), \delta(t_2))$ when $\mathsf{f} \in \{\mathsf{enc},\ \langle\ \rangle\}$.

Any tagged protocol is actually type-compliant w.r.t. its induced typing system.

Proposition 6 *Let P be a tagged protocol and let $(\mathcal{T}_P, \delta_P)$ as defined in Definition 19.*

1. *$(\mathcal{T}_P, \delta_P)$ is a typing system. We say that it is the typing system* induced by *P.*
2. *P is type-compliant w.r.t. $(\mathcal{T}_P, \delta_P)$.*

Proof. First we prove by induction on terms t and t' in Definition 5 that (T_P, δ_P) as introduced in Definition 19 is a typing system. Then we prove that if P is tagged then P is type-compliant w.r.t. (T_P, δ_P). Indeed, let $s, t \in ESt(\mathsf{unfold}^2(P))$ and a substitution σ such that $s\sigma = t\sigma$. We need to prove that $\delta_P(s) = \delta_P(t)$. Because P is tagged, $\overline{s\sigma_P} = \overline{t\sigma_P}$. Moreover, $\delta_P(s) = \delta_P(\overline{s\sigma_P})$ and $\delta_P(t) = \delta_1(\overline{t\sigma_P})$. Hence $\delta_P(s) = \delta_P(\overline{s\sigma_P}) = \delta_P(\overline{t\sigma_P}) = \delta_P(t)$. □

Corollary 1 *The problem of deciding whether two simple and tagged protocols P and Q are trace equivalent (*i.e. *$P \approx Q$) is decidable.*

Proof. Since P (resp. Q) is tagged, thanks to Proposition 6, we know that P (resp. Q) is type-compliant w.r.t. $(\mathcal{T}_P, \delta_P)$ (resp. $(\mathcal{T}_Q, \delta_Q)$), the typing system associated to P (resp. Q) as defined in Definition 19. With such typing systems, we have that the size of a term (*i.e.* number of function symbols) is smaller that the size "indicated" by its type (*i.e.* the size of the type, viewed as a term). Thus, it is then easy to see that the set:

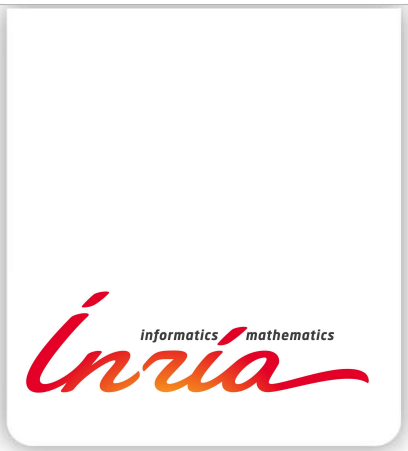
$$\{t \in \mathcal{T}(\Sigma_{\mathsf{c}}, \mathsf{A}) \ \mid \ \delta(t) = \tau\}$$

is finite for any $\tau \in \mathcal{T}_P$ (and similarly for Q) as soon as A is a set of names and constants that contains only a finite number of names/constants of each type. We conclude by applying Theorem 2. □

Finally, for the special case of strongly tagged protocols, we provide Lemma 3.

Lemma 3 *Let P be a protocol. If P is strongly tagged then P is tagged.*

Proof. Let us assume P is a strongly tagged protocol and σ_P be the substitution as in Definition 10. Let $s_1, s_2 \in ESt(\mathsf{unfold}^2(P))$ such that there exists σ with $s_1\sigma = s_2\sigma$. As both terms are of the form $\mathsf{enc}(\langle \mathsf{c}, u_i \rangle, v_i)$ for some u_i and v_i and are unifiable, they share the same tagging constant c. Then $s_1\sigma_P = s_2\sigma_P$ by Definition 10, and thus, *a fortiori*, $\overline{s_1\sigma_1} = \overline{s_2\sigma_2}$. □



RESEARCH CENTRE
NANCY – GRAND EST

615 rue du Jardin Botanique
CS20101
54603 Villers-lès-Nancy Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399