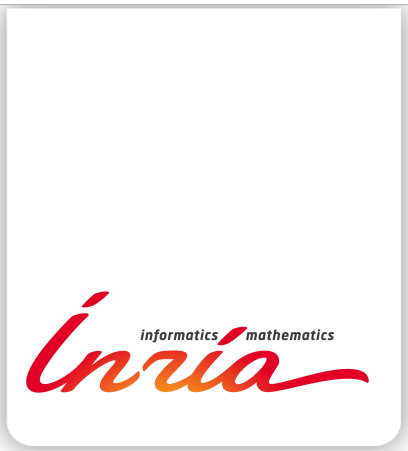
From security protocols to pushdown automata

Rémy Chrétien, Véronique Cortier, Stéphanie Delaune

RESEARCH REPORT
N° 8290
April 24, 2013
Project-Team Cassis

ISSN 0249-6399 ISRN INRIA/RR--8290--FR+ENG



From security protocols to pushdown automata *

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

Project-Team Cassis

Research Report n° 8290 — April 24, 2013 — 36 pages

Abstract: Formal methods have been very successful in analyzing security protocols for reachability properties such as secrecy or authentication. In contrast, there are very few results for equivalence-based properties, crucial for studying e.g. privacy-like properties such as anonymity or vote secrecy.
We study the problem of checking equivalence of security protocols for an unbounded number of sessions. Since replication leads very quickly to undecidability (even in the simple case of secrecy), we focus on a limited fragment of protocols (standard primitives but pairs, one variable per protocol's rules) for which the secrecy preservation problem is known to be decidable. Surprisingly, this fragment turns out to be undecidable for equivalence. Then, restricting our attention to deterministic protocols, we propose the first decidability result for checking equivalence of protocols for an unbounded number of sessions. This result is obtained through a characterization of equivalence of protocols in terms of equality of languages of (generalized, real-time) deterministic pushdown automata.

Key-words: formal methods, cryptographic protocols, pushdown automata, 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 & INRIA project Secsi & ENS Cachan
‡ LORIA, CNRS & INRIA project Cassis

RESEARCH CENTRE NANCY – GRAND EST

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

Des protocoles de sécurité aux automates à pile

Résumé : Les méthodes formelles ont rencontré beaucoup de succès dans l'analyse de protocoles de sécurité pour des propriétés d'accessibilité telles que le secret ou l'authentification. À l'inverse, il existe très peu de résultats pour des propriétés basées sur l'équivalence, cruciales pour étudier par exemple des propriétés comme l'anonymat ou le secret du vote.

Nous étudions le problème de la vérification de l'équivalence de protocoles sécurisés pour un nombre non borné de sessions. Dans la mesure où la réplication conduit très vite à l'indécidabilté (même dans le cas simple du secret), nous nous restreignons à un fragment limité de protocoles (primitives standards sans paires, une variable par règle du protocole) pour lequel le problème du secret est décidable. De façon surprenante, l'équivalence dans ce fragment s'avère être indécidable. Par la suite, en nous concentrant sur les protocoles déterministes, nous proposons le premier résultat de décidabilité pour la vérification de l'équivalence de protocoles pour un nombre non borné de sessions. Ce résultat est obtenu via une caractérisation de l'équivalence de protocoles en termes d'égalité de langages d'automates à pile (généralisés, temps-réel).

Mots-clés : méthodes formelles, protocoles cryptographiques, automates à pile, équivalence

1 Introduction

Formal methods have been successfully applied for rigorously analyzing security protocols. In particular, many algorithms and tools (see [**?**, **?**, **?**, **?**, **?**] to cite a few) have been designed to automatically find flaws in protocols or prove security. Most of these results focus on reachability properties such as authentication or secrecy: for any execution of the protocol, it should never be the case that an attacker learns some secret (secrecy property) or that an attacker makes Alice think she's talking to Bob while Bob did not engage a conversation with her (authentication property). However, privacy properties such as vote secrecy, anonymity, or untraceability cannot be expressed as reachability properties. They are instead defined as indistinguishability properties in [**?**, **?**]). For example, Alice's identity remains private if an attacker cannot distinguish a session where Alice is talking from a session where Bob is talking.

Studying indistinguishability properties for security protocols amounts into checking a behavioral equivalence between processes. Processes represent protocols and are specified in some process algebras such as CSP or the pi-calculus, except that messages are no longer atomic actions but terms, in order to faithfully represent cryptographic messages. Of course, considering terms instead of atomic actions considerably increases the difficulty of checking equivalence. As a matter of fact, there are just a few results for checking equivalence of processes that manipulate terms.

- Based on a procedure developed by M. Baudet [**?**], it has been shown that trace equivalence is decidable for deterministic processes with no else branches, and for a family of equational theories that captures most standard primitives [**?**]. A simplified proof of [**?**] has been proposed by Y. Chevalier and M. Rusinowitch [**?**].

- A. Tiu and J. Dawson[**?**] have designed and implemented a procedure for open bisimulation, a notion of equivalence stronger than the standard notion of trace equivalence. This procedure only works for a limited class of processes.

- V. Cheval *et al.* [**?**] have proposed and implemented a procedure for trace equivalence, and for a quite general class of processes. They consider non deterministic processes that use standard primitives, and that may involve else branches.

However, these decidability results analyse equivalence for a *bounded number of sessions* only, that is assuming that protocols are executed a limited number of times. This is of course a strong limitation. Even if no flaw is found when a protocol is executed n times, there is absolutely no guarantee that the protocol remains secure when it is executed $n+1$ times. And actually, the existing tools for a bounded number of sessions can only analyse protocols for a very limited number of sessions, typically 2 or 3. Another approach consists in implementing a procedure that is not guaranteed to terminate. This is in particular the case of ProVerif [**?**], a well-established tool for checking security of protocols. ProVerif is able to check equivalence although it does not always succeed [**?**]. Of course, Proverif does not correspond to any decidability result.

Our contribution. We study the decidability of equivalence of security protocols for an unbounded number of sessions. Even in the case of reachability properties such as secrecy, the problem is undecidable in general. We therefore focus on a class of protocols for which secrecy is decidable [**?**]. This class typically assumes that each protocol rule manipulates at most one variable. Surprisingly, even a fragment of this class (with only symmetric encryption) turns out to be undecidable for equivalence properties. We consequently further assume our protocols to be deterministic (that is, given an input, there is at most one possible output). We show that equivalence is decidable for an unbounded number of sessions and for protocols with randomized symmetric encryption (no pair, no signature, etc.). Interestingly, we show that checking for equivalence of protocols actually amounts into checking equality of languages of deterministic pushdown automata. The decidability of equality of languages of deterministic pushdown automata is a difficult problem, shown to be decidable at Icalp in 1997 [**?**]. We actually characterize equivalence of protocols in terms of equivalence of deterministic generalized real-time pushdown automata, that is deterministic pushdown automata with no epsilon-transition but such that the automata may unstack several symbols at a time. More precisely, we show how to associate to a process P an automata $\mathcal{A}_P$ such that two processes are equivalent if, and only if, their corresponding automata yield the same language and, reciprocally, we show how to associate to an automata $\mathcal{A}$ a process $P_{\mathcal{A}}$ such that two automata yield the same language if, and only if, their corresponding processes are equivalent, that is:

$$P \approx Q \Leftrightarrow L(\mathcal{A}_P) = L(\mathcal{A}_Q), \text{ and } L(\mathcal{A}) = L(\mathcal{B}) \Leftrightarrow P_{\mathcal{A}} \approx P_{\mathcal{B}}.$$

Therefore, checking for equivalence of protocols is as difficult as checking equivalence of deterministic generalized real-time pushdown automata.

2 Model for security protocols

Security protocols are modeled through a process algebra that manipulates terms.

2.1 Syntax

Term algebra. As usual, messages are represented by terms. More specifically, we consider a *sorted signature* with six sorts rand, key, msg, SimKey, PrivKey and PubKey that represent respectively random numbers, keys, messages, symmetric keys, private keys and public keys. We assume that msg subsumes the five other sorts, key subsumes SimKey, PrivKey and PubKey. We consider six function symbols senc and sdec, aenc and adec, sign and check that represent symmetric, asymmetric encryption and decryption as well as signatures. Since we are interested in the analysis of indistinguishability properties, we consider a randomized encryption scheme:

$$\begin{array}{rlrlrlrl} \mathsf{senc}: & \mathsf{msg}\times\mathsf{SimKey}\times\mathsf{rand} & \to \mathsf{msg} & & \mathsf{sdec}: & \mathsf{msg}\times\mathsf{SimKey} & \to \mathsf{msg} \\ \mathsf{aenc}: & \mathsf{msg}\times\mathsf{PubKey}\times\mathsf{rand} & \to \mathsf{msg} & & \mathsf{adec}: & \mathsf{msg}\times\mathsf{PrivKey} & \to \mathsf{msg} \\ \mathsf{sign}: & \mathsf{msg}\times\mathsf{PrivKey}\times\mathsf{rand} & \to \mathsf{msg} & & \mathsf{check}: & \mathsf{msg}\times\mathsf{PubKey} & \to \mathsf{msg} \end{array}$$

We further assume an infinite set Σ_0 of *constant symbols* of sort key or msg, an infinite set $\mathcal{C}h$ of constant symbols of sort channel, two infinite sets

of *variables* $\mathcal{X}, \mathcal{W}$, and an infinite set of *names* $\mathcal{N} = \mathcal{N}_{\mathsf{pub}} \uplus \mathcal{N}_{\mathsf{prv}}$ of *names* of sort rand: $\mathcal{N}_{\mathsf{pub}}$ represents the random numbers drawn by the attacker while $\mathcal{N}_{\mathsf{prv}}$ represents the random numbers drawn by the protocol's participants. As usual, *terms* are defined as names, variables, and function symbols applied to other terms. We denote by $\mathcal{T}(\mathcal{F}, \mathcal{N}, \mathcal{X})$ the set of terms built on function symbols in $\mathcal{F}$, names in $\mathcal{N}$, and variables in $\mathcal{X}$. We simply write $\mathcal{T}(\mathcal{F}, \mathcal{N})$ when $\mathcal{X} = \emptyset$. We consider three particular signatures:

$$\Sigma_{\mathsf{pub}} = \{\mathsf{senc}, \mathsf{sdec}, \mathsf{aenc}, \mathsf{adec}, \mathsf{sign}, \mathsf{check}, \mathsf{start}\}$$
$$\Sigma^+ = \Sigma_{\mathsf{pub}} \cup \Sigma_0 \qquad \Sigma = \{\mathsf{senc}, \mathsf{aenc}, \mathsf{sign}, \mathsf{start}\} \cup \Sigma_0$$

where $\mathsf{start} \notin \Sigma_0$ is a constant symbol of sort msg. Σ_{pub} represents the functions/data available to the attacker, Σ^+ is the most general signature, while Σ models actual messages (with no failed computation). We add a bijection between elements of sort $\mathsf{PrivKey}$ and PubKey. If k is a constant of sort $\mathsf{PrivKey}$, k^{-1} will denotes its image by this function, called *inverse*. We will write the inverse function the same, so that $(\mathsf{k}^{-1})^{-1} = \mathsf{k}$. To keep homogeneous notations, we will extend this function to symmetric keys: if k is of sort SimKey, then $\mathsf{k}^{-1} = \mathsf{k}$. The relation between encryption and decryption is represented through the following rewriting rules, yielding a convergent rewrite system:

$$\mathsf{sdec}(\mathsf{senc}(x, y, z), y) \to x \qquad \mathsf{adec}(\mathsf{aenc}(x, y, z), y^{-1}) \to x$$
$$\mathsf{check}(\mathsf{sign}(x, y, z), y^{-1}) \to x$$

This rule models the fact that the decryption of a ciphertext will return the associated plaintext when the right key is used to perform decryption. We denote by $t{\downarrow}$ the *normal form* of a term $t \in \mathcal{T}(\Sigma^+, \mathcal{N}, \mathcal{X})$.

Example 1. *The term* $m = \mathsf{senc}(\mathsf{s}, \mathsf{k}, r)$ *represents an encryption of the constant* s *with the key* k *using the random* $r \in \mathcal{N}$*, whereas* $t = \mathsf{sdec}(m, \mathsf{k})$ *models the application of the decryption algorithm on* m *using* k*. We have that* $t{\downarrow} = \mathsf{s}$*.*

An attacker may build his own messages by applying functions to terms he already knows. Formally, a computation done by the attacker is modeled by a *recipe*. i.e. a term in $\mathcal{T}(\Sigma_{\mathsf{pub}}, \mathcal{N}_{\mathsf{pub}}, \mathcal{W})$. The variables in $\mathcal{W}$ intuitively refer to variables used to store messages learnt by the attacker.

Process algebra. The intended behavior of a protocol can be modelled by a *process* defined by the following grammar where $u \in \mathcal{T}(\Sigma, \mathcal{N}, \mathcal{X})$, $n \in \mathcal{N}$, and $c \in \mathcal{C}h$:

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

The process "$\mathsf{in}(c, u).P$" expects a message m of the form u on channel c and then behaves like $P\theta$ where θ is a substitution such that $m = u\theta$. The process "$\mathsf{out}(c, u).P$" emits u on channel c, and then behaves like P. The variables that occur in u will be instantiated when the evaluation will take place. The process $P \mid Q$ runs P and Q in parallel. The process $!P$ executes P some arbitrary number of times. The process $\mathsf{new}\ n.P$ invents a new name n and continues as P.

Sometimes, we will omit the null process. 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. A *protocol* is a ground process, *i.e.* a process P such that $fv(P) = \emptyset$.

Example 2. *For the sake of illustration, we consider a naive protocol, where A sends a value v (e.g. a vote) to B, encrypted by a short-term key exchanged through a server.*

$$\begin{array}{lll} 1. & A \to S: & \mathsf{senc}(\mathsf{k}_{AB}, \mathsf{k}_{AS}, r_A) \\ 2. & S \to B: & \mathsf{senc}(\mathsf{k}_{AB}, \mathsf{k}_{BS}, r_S) \\ 3. & A \to B: & \mathsf{senc}(v, \mathsf{k}_{AB}, r) \end{array}$$

The agent A sends a symmetric key k_{AB} encrypted with the key k_{AS} (using a fresh random number r_A). The server answers to this request by decrypting this message and encrypting it with k_{BS}. The agent A can now send his vote v encrypted with k_{AB}.

The role of A is modeled by a process $P_A(v)$ while the role of S is modeled by P_S. The role of B (which does not output anything) is omitted for concision.

$$\begin{array}{rcll} P_A(v) & \stackrel{\mathsf{def}}{=} & !\,\mathsf{in}(c_A, \mathsf{start}).\mathsf{new}\, r_A.\mathsf{out}(c_A, \mathsf{senc}(\mathsf{k}_{AB}, \mathsf{k}_{AS}, r_A)) & (1) \\ & & |\,!\,\mathsf{in}(c'_A, \mathsf{start}).\mathsf{new}\, r.\mathsf{out}(c_A, \mathsf{senc}(v, \mathsf{k}_{AB}, r)) & (2) \\ P_S & \stackrel{\mathsf{def}}{=} & !\,\mathsf{in}(c_S, \mathsf{senc}(x, \mathsf{k}_{AS}, z)).\mathsf{new}\, r_S.\mathsf{out}(c_S, \mathsf{senc}(x, \mathsf{k}_{BS}, r_S)) & (3) \\ & & |\,!\,\mathsf{in}(c'_S, \mathsf{senc}(x, \mathsf{k}_{AS}, z)).\mathsf{new}\, r_S.\mathsf{out}(c'_S, \mathsf{senc}(x, \mathsf{k}_{CS}, r_S)) & (4) \end{array}$$

where c_A, c'_A, c_S, c'_S are constants of sort channel*, k_{AB}, k_{AS}, k_{BS}, and k_{CS} are (private) constants in Σ_0 of sort* key*, whereas r_A, r_S, r are names of sort* rand*, and x (resp. z) is a variable of sort* msg *(resp.* rand*).*

Intuitively, $P_A(v)$ sends k_{AB} encrypted by k_{AS} to the server (branch 1), and then her vote encrypted by k_{AB} (branch 2). The process P_S models the server, answering both requests from A to B (branch 3), as well as requests from A to C (branch 4). More generally the server answers requests from any agent to any agent but only two cases are considered here, again for concision. The whole protocol is given by $P(v)$, where $P_A(v)$ and P_S evolve in parallel and additionally, the secret key k_{CS} is sent in clear, to model the fact that the attacker may learn keys of some corrupted agents:

$$P(v) \stackrel{\mathsf{def}}{=} P_A(v) \;\mid\; P_S \;\mid !\,\mathsf{in}(c, \mathsf{start}).\mathsf{out}(c, \mathsf{k}_{CS})$$

2.2 Semantics

A *configuration* of a protocol is a pair $(\mathcal{P}; \sigma)$ where:

- $\mathcal{P}$ is a multiset of processes. We often write $P \cup \mathcal{P}$, or $P \mid \mathcal{P}$, instead of $\{P\} \cup \mathcal{P}$.
- $\sigma = \{\mathsf{w}_1 \triangleright m_1, \ldots, \mathsf{w}_n \triangleright m_n\}$ 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 terms in $\mathcal{T}(\Sigma, \mathcal{N})$. Those terms represent the messages that are known by the attacker.

The operational semantics of protocol is defined by the relation $\xrightarrow{\alpha}$ over configurations. For sake of simplicity, we often write P instead of $(P; \emptyset)$.

$(\mathsf{in}(c,u).P \cup \mathcal{P}; \sigma) \xrightarrow{\mathsf{in}(c,R)} (P\theta \cup \mathcal{P}; \sigma)$
where R is a recipe such that $R\sigma\downarrow \in \mathcal{T}(\Sigma,\mathcal{N})$ and $R\sigma\downarrow = u\theta$ for some θ

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

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

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

A process may input any term that an attacker can build (rule In). The process $\mathsf{out}(c,u).P$ outputs u (which is stored in the attacker's knowledge) and then behaves like P. The two remaining rules are unobservable (τ action) from the point of view of the attacker. The relation $\xrightarrow{w}$ between configurations (where w is a sequence of actions) is defined in a usual way. Given a sequence of observable actions w, we write $K \stackrel{w}{\Longrightarrow} K'$ when there exists w' such that $K \xrightarrow{w'} K'$ and w is obtained from w' by erasing all occurrences of τ. For every configuration K, we define its *set of traces* as follows:

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

Example 3. *Going back to the protocol introduced in Example 2, consider the following scenario:* (i) *the corrupted agent C discloses his secret key k_{CS};* (ii) *the agent A initiates a session with B, and for this she sends a request to the server S;* (iii) *the attacker intercepts this message and sends it to S as a request coming from A to establish a key with C. Instead of answering to this request with $\mathsf{senc}(\mathsf{k}_{AB}, \mathsf{k}_{BS}, r_S)$, the server sends $\mathsf{senc}(\mathsf{k}_{AB}, \mathsf{k}_{CS}, r_S)$, and the attacker will learn k_{AB}. More formally, we have that:*

$$K_0 \stackrel{\mathsf{def}}{=} (P(v); \emptyset) \xrightarrow{\mathsf{in}(c,\mathsf{start}).\mathsf{out}(c,\mathsf{w}_1).\mathsf{in}(c_A,\mathsf{start}).\mathsf{out}(c_A,\mathsf{w}_2).\mathsf{in}(c'_S,\mathsf{w}_2).\mathsf{out}(c'_S,\mathsf{w}_3).} (P(v); \sigma)$$

where $\sigma = \{\mathsf{w}_1 \triangleright \mathsf{k}_{CS},\ \mathsf{w}_2 \triangleright \mathsf{senc}(\mathsf{k}_{AB}, \mathsf{k}_{AS}, r_A),\ \mathsf{w}_3 \triangleright \mathsf{senc}(\mathsf{k}_{AB}, \mathsf{k}_{CS}, r_S)\}$, and r_A, r_S are (fresh) names in $\mathcal{N}_{\mathsf{prv}}$. In this execution trace, first the key k_{CS} is sent after having called the corresponding process. Then, branches (1) *and* (4) *of $P(v)$ are triggered.*

2.3 Trace equivalence

Intuitively, two processes 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 [**?**, **?**]. We first introduce a notion of intruder's knowledge well-suited to cryptographic primitives for which the success of decrypting or checking a signature is visible.

Definition 1. *Two frames σ_1 and σ_2 are* statically equivalent, *$\sigma_1 \sim \sigma_2$, when we have that $dom(\sigma_1) = dom(\sigma_2)$, and:*

- *for any recipe R, $R\sigma_1\downarrow \in \mathcal{T}(\Sigma,\mathcal{N})$ if, and only if, $R\sigma_2\downarrow \in \mathcal{T}(\Sigma,\mathcal{N})$; and*
- *for all recipes R_1 and R_2 such that $R_1\sigma_1\downarrow, R_2\sigma_1\downarrow \in \mathcal{T}(\Sigma,\mathcal{N})$, we have that $R_1\sigma_1\downarrow = R_2\sigma_1\downarrow$ if, and only if, $R_1\sigma_2\downarrow = R_2\sigma_2\downarrow$.*

Intuitively, two frames are equivalent if an attacker cannot see the difference between the two situations they represent: if some computation fails in σ_1 it should fail in σ_2 as well, and σ_1 and σ_2 should satisfy the same equalities.

Example 4. *Assume some agent publishes her vote encrypted. The possible values for the votes are typically public. Therefore the question is not whether an attacker may know the value of the vote (that he knows anyway) but instead, whether he may distinguish between two executions where A votes differently. Consider the two frames:*

$$\sigma_i \stackrel{\mathsf{def}}{=} \{\mathsf{w}_4 \rhd \mathsf{v}_0,\ \mathsf{w}_5 \rhd \mathsf{v}_1,\ \mathsf{w}_6 \rhd \mathsf{senc}(\mathsf{v}_i, \mathsf{k}_{AB}, r)\} \text{ with } i \in \{0,1\}$$

where $\mathsf{v}_0, \mathsf{v}_1 \in \Sigma_0$*, and* $r \in \mathcal{N}_{\mathsf{prv}}$*. We have that* $\sigma_0 \sim \sigma_1$*. Intuitively, there is no test that allows the attacker to distinguish the two frames since the key* k_{AB} *is not available. In this scenario, the vote* v_i *remains private. Now, consider the frames* $\sigma'_i = \sigma \cup \sigma_i$ *with* $i \in \{0,1\}$ *and* σ *as defined in Example 3. We have that* $\sigma'_0 \not\sim \sigma'_1$*. Indeed, consider the recipes* $R_1 = \mathsf{sdec}(\mathsf{w}_6, \mathsf{sdec}(\mathsf{w}_3, \mathsf{w}_1))$ *and* $R_2 = \mathsf{w}_4$*. We have that* $R_1\sigma'_0{\downarrow} = R_2\sigma'_0{\downarrow} = \mathsf{v}_0$*, whereas* $R_1\sigma'_1{\downarrow} = \mathsf{v}_1$ *and* $R_2\sigma'_1{\downarrow} = \mathsf{v}_0$*. Intuitively, an attacker can learn* k_{AB} *and then compare the encrypted vote to the values* v_0 *and* v_1*.*

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

Definition 2. *Let* P *and* Q *be two protocols. We have that* $P \sqsubseteq Q$ *if for every* $(\mathsf{tr}, \sigma) \in \mathsf{trace}(P)$*, there exists* $(\mathsf{tr}', \sigma') \in \mathsf{trace}(Q)$ *such that* $\mathsf{tr} = \mathsf{tr}'$ *and* $\sigma \sim \sigma'$*. They are* trace equivalent*, written* $P \approx Q$*, if* $P \sqsubseteq Q$ *and* $Q \sqsubseteq P$*.*

Example 5. *Continuing Example 2, our naive protocol is secure if the vote of A remains private. This is typically expressed by* $P(\mathsf{v}_0) \mid Q \approx P(\mathsf{v}_1) \mid Q$*. An attacker should not distinguish between two instances of the protocol where A votes two different values. The purpose of* Q *is to disclose the two values* v_0 *and* v_1*.*

$$Q \stackrel{\mathsf{def}}{=} !\,\mathsf{in}(c_0, \mathsf{start}).\mathsf{out}(c_0, \mathsf{v}_0) \mid !\,\mathsf{in}(c_1, \mathsf{start}).\mathsf{out}(c_1, \mathsf{v}_1)$$

However, our protocol is insecure. As seen in Example 3, an attacker may learn k_{AB}*, and therefore distinguish between the two processes described above. Formally, we have that* $P(\mathsf{v}_0) \mid Q \not\approx P(\mathsf{v}_1) \mid Q$*. This is reflected by the trace* tr' *described below:*

$$\mathsf{tr}' \stackrel{\mathsf{def}}{=} \mathsf{tr}.\mathsf{in}(c_0, \mathsf{start}).\mathsf{out}(c_0, \mathsf{w}_4).\mathsf{in}(c_1, \mathsf{start}).\mathsf{out}(c_1, \mathsf{w}_5).\mathsf{in}(c'_A, \mathsf{start}).\mathsf{out}(c'_A, \mathsf{w}_6).$$

We have that $(\mathsf{tr}', \sigma'_0) \in \mathsf{trace}(K_0)$ *with* $K_0 = (P(\mathsf{v}_0) \mid Q; \emptyset)$ *and* σ'_0 *as defined in Example 4. Because of the existence of only one branch using each channel, there is only one possible execution of* $P(\mathsf{v}_1) \mid Q$ *(up to a bijective renaming of the private names of sort* rand*) matching the labels in* tr'*, and the corresponding execution will allow us to reach the frame* σ'_1 *as described in Example 4. We have already seen that static equivalence does not hold, i.e.* $\sigma'_0 \not\sim \sigma'_1$*.*

3 Ping-pong protocols

We aim at providing a decidability result for the problem of trace equivalence between protocols in presence of replication. However, it is well-known that

replication leads to undecidability even for the simple case of reachability properties. Thus, we consider a class of protocols, called $\mathcal{C}_{\mathsf{pp}}$, for which (in a slightly different setting), reachability has already been proved decidable [**?**].

3.1 Class $\mathcal{C}_{\mathsf{pp}}$

We basically consider ping-pong protocols (an output is computed using only the message previously received in input), and we assume a kind of determinism. Moreover, we restrict the terms that are manipulated throughout the protocols: only one unknown message (modelled by the use of a variable of sort msg) can be received at each step.

We fix a variable $x \in \mathcal{X}$ of sort msg. An *input term* (resp. *output term*) is a term defined by the grammars given below:

$$u := x \mid \mathsf{s} \mid \mathsf{f}(u, \mathsf{k}, z) \qquad\qquad v := x \mid \mathsf{s} \mid \mathsf{f}(v, \mathsf{k}, r)$$

where $\mathsf{s}, \mathsf{k} \in \Sigma_0 \cup \{\mathsf{start}\}$, $z \in \mathcal{X}$, $\mathsf{f} \in \{\mathsf{senc}, \mathsf{aenc}, \mathsf{sign}\}$ and $r \in \mathcal{N}$. Moreover, we assume that each variable (resp. name) occurs at most once in u (resp. v).

Definition 3. *$\mathcal{C}_{\mathsf{pp}}$ is the class of protocol of the form:*

$$P = \prod_{i=1}^{n} \prod_{j=1}^{p_i} !\mathsf{in}(c_i, u_j^i).\mathsf{new}\ r_1.\ldots.\mathsf{new}\ r_{k_j^i}.\mathsf{out}(c_i, v_j^i) \quad \textit{such that:}$$

1. *for all $i \in \{1, \ldots, n\}$, and $j \in \{1, \ldots, p_i\}$, $k_j^i \in \mathbb{N}$, u_j^i is an input term, and v_j^i is an output term where names occurring in v_j^i are included in $\{r_1, \ldots, r_{k_j^i}\}$;*

2. *for all $i \in \{1, \ldots, n\}$, and $j_1, j_2 \in \{1, \ldots, p_i\}$, if $j_1 \neq j_2$ then for any renaming of variables, $u_{j_1}^i$ and $u_{j_2}^i$ are not unifiable*[1].

Note that the purpose of item *2* is to restrict the class of protocols to those that have a deterministic behavior (a particular input action can only be accepted by one branch of the protocol). This is a natural restriction since most of the protocols are indeed deterministic: an agent should usually know exactly what to do once he has received a message. Actually, the main limitations of the class $\mathcal{C}_{\mathsf{pp}}$ are stated in item *1*: we consider a restricted signature (e.g. no pair, no hash function), and names can only be used to produce randomized ciphertexts.

Example 6. *The protocols described in Example 5 are in $\mathcal{C}_{\mathsf{pp}}$. For instance, we can check that $\mathsf{senc}(x, \mathsf{k}_{AS}, z)$ is an input term whereas $\mathsf{senc}(x, \mathsf{k}_{BS}, r_S)$ is an output term. Moreover, the determinism condition (item 2) is clearly satisfied: each branch of the protocol $P(\mathsf{v}_0) \mid Q$ (resp. $P(\mathsf{v}_0) \mid Q$) uses a different channel.*

Our main contribution is a decision procedure for trace equivalence of processes in $\mathcal{C}_{\mathsf{pp}}$. Details of the procedure are provided in Section 4.

Theorem 1. *Let P and Q be two protocols in $\mathcal{C}_{\mathsf{pp}}$. The problem whether P and Q are trace equivalent, i.e. $P \approx Q$, is decidable.*

[1] i.e. there does not exist θ such that $u_{j_1}^i\theta = u_{j_2}^i\theta$.

3.2 Undecidability results

The class $\mathcal{C}_{\mathsf{pp}}$ is somewhat limited but surprisingly, extending $\mathcal{C}_{\mathsf{pp}}$ to non deterministic processes immediately yields undecidability of trace equivalence. More precisely, trace inclusion of processes in $\mathcal{C}_{\mathsf{pp}}$ is already undecidable.

Theorem 2. *Let P and Q be two protocols in $\mathcal{C}_{\mathsf{pp}}$. The problem whether P is trace included in Q, i.e. $P \sqsubseteq Q$, is undecidable.*

This result is shown by encoding the Post Correspondence Problem (PCP). Alternatively, it results from the reduction result established in Section 5 and the undecidability result established in [**?**]. Undecidability of trace inclusion actually implies undecidability of trace equivalence as soon as processes are non deterministic. Indeed consider the choice operator $+$ whose (standard) semantics is given by the following rules:

$$(\{P + Q\} \cup \mathcal{P}; \sigma) \xrightarrow{\tau} (P \cup \mathcal{P}; \sigma) \qquad (\{P + Q\} \cup \mathcal{P}; \sigma) \xrightarrow{\tau} (Q \cup \mathcal{P}; \sigma)$$

Corollary 1. *Let P, Q_1, and Q_2 be three protocols in $\mathcal{C}_{\mathsf{pp}}$. The problem whether P is equivalent to $Q_1 + Q_2$, i.e. $P \approx Q_1 + Q_2$, is undecidable.*

Indeed, consider P and Q_1, for which trace inclusion encodes PCP, and let $Q_2 = P$. Trivially, $P \sqsubseteq Q_1 + Q_2$. Thus $P \approx Q_1 + Q_2$ if, and only if, $Q_1 + Q_2 \sqsubseteq P$, i.e. if, and only if, $Q_1 \sqsubseteq P$, hence the undecidability result.

4 From trace equivalence to language equivalence

This section is devoted to a sketch of proof of Theorem 1. Deciding trace equivalence is done in two main steps. First, we show how to reduce the trace equivalence problem between protocols in $\mathcal{C}_{\mathsf{pp}}$, to the problem of deciding trace equivalence (still between protocols in $\mathcal{C}_{\mathsf{pp}}$) when the attacker acts as a *forwarder*. Then, we encode the problem of deciding trace equivalence for forwarding attackers into the problem of language equivalence for real-time generalized pushdown deterministic automata (GPDA).

4.1 Generalized pushdown automata

GPDA differ from deterministic pushdown automata (DPA) as they can unstack several symbols at a time. We consider real-time GPDA with final-state acceptance.

Definition 4. *A* real-time GPDA *is a 7-tuple $\mathcal{A} = (Q, \Pi, \Gamma, q_0, \omega, Q_f, \delta)$ where Q is the finite set of states, $q_0 \in Q$ is the initial state, $Q_f \subseteq Q$ is the set of accepting states, Π is the finite input-alphabet, Γ is the finite stack-alphabet, ω is the initial stack symbol, and $\delta : (Q \times \Pi \times \Gamma_0) \to Q \times \Gamma_0$ is the partial transition function such that:*

- *Γ_0 is a finite subset of Γ^*; and*
- *for any $(q, a, x) \in dom(\delta)$ and y suffix strict of x, we have that $(q, a, y) \notin dom(\delta)$.*

Let $q, q' \in Q$, $w, w', \gamma \in \Gamma^*$, $m \in \Pi^*$, $a \in \Pi$; we note $(qw\gamma, am) \rightsquigarrow_{\mathcal{A}} (q'ww', m)$ if $(q', w') = \delta(q, a, \gamma)$. The relation $\rightsquigarrow_{\mathcal{A}}^*$ is the reflexive and transitive closure of $\rightsquigarrow_{\mathcal{A}}$. For every qw, $q'w'$ in $Q\Gamma^*$ and $m \in \Pi^*$, we note $qw \xrightarrow{m}_{\mathcal{A}} q'w'$ if, and only if, $(qw, m) \rightsquigarrow_{\mathcal{A}}^* (q'w', \epsilon)$. For sake of clarity, a transition from q to q' reading a, popping γ from the stack and pushing w' will be denoted by $q \xrightarrow{a;\gamma/w'} q'$.

Let $\mathcal{A}$ be a GPDA. The language recognized by $\mathcal{A}$ is defined by:

$$\mathcal{L}(\mathcal{A}) = \{m \in \Pi^* \mid q_0\omega \xrightarrow{m}_{\mathcal{A}} q_f w \text{ for some } q_f \in Q_f \text{ and } w \in \Gamma^*\}.$$

A real-time GPDA can easily be converted into a DPA by adding new states and ϵ-transitions. Thus, the problem of language equivalence for two real-time GPDA $\mathcal{A}_1$ and $\mathcal{A}_2$, i.e. deciding whether $\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2)$ is decidable [**?**].

4.2 Getting rid of the attacker

We define the actions of a forwarder by modifying our semantics. We restrict the recipes R, R_1, and R_2 that are used in the In rule and in static equivalence (Definition 1) to be either the public constant start or a variable in $\mathcal{W}$. This leads us to consider a new relation $\Rightarrow_{\mathsf{fwd}}$ between configurations, and a new notion of static equivalence $\sim_{\mathsf{fwd}}$. We denote by $\approx_{\mathsf{fwd}}$ the trace equivalence relation induced by this new semantics.

Example 7. *The trace exhibited in Example 3 is still a valid one according to the forwarder semantics, and the frames σ_0' and σ_1' described in Example 4 are in equivalence according to $\sim_{\mathsf{fwd}}$. Actually, we have that $P(\mathsf{v}_0) \mid Q \approx_{\mathsf{fwd}} P(\mathsf{v}_1) \mid Q$. Indeed, the fact that a forwarder simply acts as a relay prevents him to mount the aforementioned attack.*

As shown above, the forwarder semantics is very restrictive: a forwarder can not rely on his deduction capabilities to mount an attack. To counterbalance the effects of this semantics, the key idea consists in modifying the protocols under study by adding new rules that encrypt and decrypt messages on demand for the forwarder.

Formally, we define a transformation $\mathcal{T}_{\mathsf{fwd}}$ that associates to a pair of protocols in $\mathcal{C}_{\mathsf{pp}}$ a finite set of pairs of protocols (still in $\mathcal{C}_{\mathsf{pp}}$), and we show the following result:

Proposition 1. *Let P and Q be two protocols in $\mathcal{C}_{\mathsf{pp}}$. We have that:*

$$P \approx Q \text{ if, and only if, } P' \approx_{\mathsf{fwd}} Q' \text{ for some } (P', Q') \in \mathcal{T}_{\mathsf{fwd}}(P, Q).$$

Roughly the transformation $\mathcal{T}_{\mathsf{fwd}}$ consists in first guessing among the keys of the protocols P and the keys of the protocols Q those that are deducible by the attacker, as well as a bijection α between these two sets. We can show that such a bijection necessarily exists when $P \approx Q$. Then, to compensate the fact that the attacker is a simple forwarder, we give him access to encryption/decryption oracles for any deducible key k, adding the corresponding branches in the processes, i.e.:

$$!\,\mathsf{in}(c_{\mathsf{k}}^{\mathsf{senc}}, x).\mathsf{new}\, r.\mathsf{out}(c_{\mathsf{k}}^{\mathsf{senc}}, \mathsf{senc}(x, \mathsf{k}, r)) \;\mid !\,\mathsf{in}(c_{\mathsf{k}}^{\mathsf{sdec}}, \mathsf{senc}(x, \mathsf{k}, z)).\mathsf{out}(c_{\mathsf{k}}^{\mathsf{sdec}}, x)$$

To maintain the equivalence, we do a similar transformation in both P and Q relying on the bijection α. We ensure that the set of deducible keys has been correctly guessed by adding of some extra processes. Then the main step of

the proof consists in showing that the forwarder has now the same power as a full attacker, although he cannot reuse the same randomness in two distinct encryptions, as a real attacker could.

4.3 Encoding a protocol into a real-time GPDA

For any process $P \in \mathcal{C}_{\mathsf{pp}}$, we can show that it is possible to define a polynomial-sized a real-time GPDA $\mathcal{A}_P$ such that trace equivalence against forwarder of two processes coincides with language equivalence of the two corresponding automata.

Theorem 3. *Let P and Q in $\mathcal{C}_{\mathsf{pp}}$, we have that: $P \approx_{\mathsf{fwd}} Q \iff \mathcal{L}(\mathcal{A}_P) = \mathcal{L}(\mathcal{A}_Q)$.*

The idea is that the automaton $\mathcal{A}_P$ associated to a protocol P recognizes the words (a sequence of channels) that correspond to a possible execution in P. The stack of $\mathcal{A}_P$ is used to store a (partial) representation of the last outputted term. This requires to convert a term into a word, and we use the following representation:

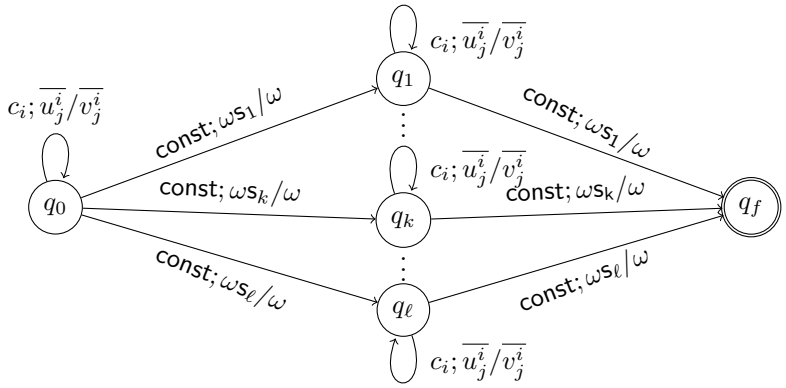
$\bar{\mathsf{s}} = \mathsf{s}$ for any constant $\mathsf{s} \in \Sigma_0 \cup \{\mathsf{start}\}$; and $\overline{\mathsf{f}(v, \mathsf{k}, r)} = \bar{v}.\mathsf{k}$ otherwise.

Note that, even if our signature is infinite, we show that only a finite number of constants of sort msg and a finite number of constants of sort $\mathsf{channel}$ need to be considered (namely those that occur in the protocols under study). Thus, the stack-alphabet and the input-alphabet of the automaton are both finite.

To construct the automaton associated to a process $P \in \mathcal{C}_{\mathsf{pp}}$, we need to construct an automaton that recognizes any execution of P and the corresponding valid tests. For the sake of illustration, we present only the automaton (depicted below) that recognizes tests of the form $\mathsf{w} = \mathsf{w}'$ such that the corresponding term is actually a constant.

Intuitively, the basic building blocks (e.g. q_0 with the transitions from q_0 to itself) mimic an execution of P where each input is fed with the last outputted term. Then, to recognize the tests of the form $\mathsf{w} = \mathsf{w}'$ that are true in such an execution, it is sufficient to memorize the constant s_i that is associated to w (adding a new state q_i), and to see whether it is possible to reach a state where the stack contains s_i again.

Capturing tests that lead to non-constant symbols (i.e. terms of the form $\mathsf{senc}(u, \mathsf{k}, r)$) is more tricky for several reasons. First, it is not possible anymore to memorize the resulting term in a state of the automaton. Second, names of sort rand play a role in such a test, while they are forgotten in our encoding. We therefore have to, first, characterize more precisely trace equivalence and secondly, construct more complex automata that use some special track symbols to encode when randomized ciphertexts may be reused.



5 From language equivalence to trace equivalence

We have just seen how to encode equivalence of processes in $\mathcal{C}_{\mathsf{pp}}$ into real-time GPDA. The equivalence of processes in $\mathcal{C}_{\mathsf{pp}}$ is actually *equivalent* to language equivalence of real-time GPDA. Indeed, we can conversely encode any a real-time GPDA into a process in $\mathcal{C}_{\mathsf{pp}}$, preserving equivalence. The transformation works as follows.

Given a word $\alpha_1 \ldots . \alpha_p$, for sake of concision, the expression $x.u$ will denote either the term $\mathsf{senc}(\ldots \mathsf{senc}(x, \alpha_1, z_1), \ldots), \alpha_p, z_p)$ when it occurs as an input term; or $\mathsf{senc}(\ldots \mathsf{senc}(x, \alpha_1, r_1), \ldots), \alpha_p, r_p)$ when it occurs as an output term. Then given an automaton $\mathcal{A} = (Q, \Pi, \Gamma, q_0, \omega, Q_f, \delta)$, the corresponding process $P_{\mathcal{A}}$ is defined as follows:

$$
\begin{array}{rcl}
P_{\mathcal{A}} \stackrel{\mathsf{def}}{=} & & !\,\mathsf{in}(c_0, \mathsf{start}).\mathsf{new}\ r.\mathsf{out}(c_0, \mathsf{senc}(\omega, q_0, r)) \\
& | & !\,\mathsf{in}(c_a, \mathsf{senc}(x.u, q, z)).\mathsf{new}\ \tilde{r}.\mathsf{out}(c_a, \mathsf{senc}(x.v, q', r)) \\
& | & !\,\mathsf{in}(c_f, \mathsf{senc}(x, q_f, z)).\mathsf{out}(c_f, \mathsf{start}) \\
& | & P'_{\mathcal{A}}
\end{array}
$$

where a quantifies over Π, q over Q, u over words in Γ^* such that $(q, a, u) \in dom(\delta)$, q_f over Q_f, and $(q', v) = \delta(q, a, u)$.

Intuitively, the stack of the automata $\mathcal{A}$ is encoded as a pile of encryptions (where each key encodes a tile of the stack). Then, upon receiving a stack s encrypted by q on channel c_a, the process $P_{\mathcal{A}}$ mimics the transition of $\mathcal{A}$ at state q and stack s, upon reading a. The resulting stack is sent encrypted by the resulting state. This polynomial encoding (with some additional technical details hidden in $P'_{\mathcal{A}}$) preserves equivalence.

Proposition 2. *Let $\mathcal{A}$ and $\mathcal{B}$ be two real-time GPDA: $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B}) \iff P_{\mathcal{A}} \sqsubseteq P_{\mathcal{B}}$.*

Therefore, checking for equivalence of protocols is as difficult as checking equivalence of real-time generalized pushdown deterministic automata. It follows that the exact complexity of checking equivalence of protocols is unknown. The only upper bound is that equivalence is at most primitive recursive. This bound comes from the algorithm proposed by C. Stirling for equivalence of DPA [**?**] (Icalp 2002). Whether equivalence of DPA (or even real-time GPDA) is e.g. at least NP-hard is unknown.

6 Conclusion

We have shown a first decidability result for equivalence of security protocols for an unbounded number of sessions by reducing it to the equality of languages of deterministic pushdown automata. We further show that deciding equivalence of security protocols is actually at least as hard as deciding equality of languages of deterministic, generalized, real-time pushdown automata.

Our class of security protocols handles only randomized primitives, namely symmetric/asymmetric encryptions and signatures. Our decidability result could be extended to handle deterministic primitives instead of the randomized one (the reverse encoding - from real-time GPDAs to processes with deterministic encryption - may not hold anymore). Due to the use of pushdown automata, extending our decidability result to protocols with pair is not straightforward. A direction is to use pushdown automata for which stacks are terms.

G. Sénizergues is currently implementing his procedure for pushdown automata [**?**]. As soon as the tool will be available, we plan to implement our translation, yielding a tool for automatically checking equivalence of security protocols, for an unbounded number of sessions.

A Undecidability results

This section is devoted to the proof of Theorem 2 introduced in Section 3.2.

Theorem 2. *Let P and Q be two protocols in $\mathcal{C}_{\mathsf{pp}}$. The problem whether P is trace included in Q, i.e. $P \sqsubseteq Q$, is undecidable.*

To prove the undecidability of trace inclusion in $\mathcal{C}_{\mathsf{pp}}$, we show it is possible to encode the Post Correspondence Problem into an inclusion of two protocols of this class. Given a word, one protocol will be meant to unstack the first set of tiles while the other will try as much as possible to unstack the second set of tiles. While an empty word is not "simultaneously" reached by the two processes, their traces appear to be equivalent. On the other hand, if a solution to the Post Correspondence Problem does exit, it will lead the second process to react in a distinct way, breaking the trace inclusion property.

Let PCP be an instance of the Post Correspondence Problem over the alphabet A, with sets of non-empty tiles $U = \{u_i\}_{1\leq i\leq n}$ and $V = \{v_i\}_{1\leq i\leq n}$. We can then define:

$$\forall i \in \{1,\dots,n\},\ \begin{cases} W_i = A^{|v_i|} \setminus \{v_i\} \\ W'_i = \bigcup_{k=1}^{|v_i|-1} A^k \end{cases}$$

Words in A^* will be represented through nested symmetric encryption with private keys representing their counterparts in A. For the sake of brevity, given a word $u = \alpha_1 \dots \alpha_p$ of A^*, we will denote by $x.u$:

- either the term $\mathsf{senc}(\dots \mathsf{senc}(x, \alpha_1, y_1) \dots, \alpha_p, y_p)$ where y_1 through y_p are variables of sort rand when $x.u$ is used as an input pattern;
- or the term $\mathsf{senc}(\dots \mathsf{senc}(x, \alpha_1, n_1) \dots, \alpha_p, n_p)$ where n_1 through n_p are names when $x.u$ occurs in an output pattern.

We denote by $\tilde{r}$ the sequence of $\mathsf{new}\ r_i$ that binds every nonce in the following output.

Below, $\mathsf{k}_i, \mathsf{k}'_i$ with $i \in \{0,1,2,3\}$ are constants in Σ_0 of sort key, and for each $a \in A$, we denote also by a its counterpart in Σ_0 (constants of sort key). Lastly, we denote ϵ a constant in Σ_0 of sort msg, and c, c_a, c_i, c' with $a \in A$ and $i \in \{1,\dots,n\}$ are constant symbols in $\mathcal{C}h$.

Let P_U and P_V be the following protocols. Note that P_U and P_V are both in $\mathcal{C}_{\mathsf{pp}}$.

$$\begin{array}{lll}
P_U := & !\,\mathsf{in}(c, \mathsf{start}).\mathsf{new}\ r.\mathsf{out}(c, \mathsf{senc}(\epsilon, \mathsf{k}_0, r)) & (\mathsf{start}) \\
| & !\,\mathsf{in}(c_a, \mathsf{senc}(x, \mathsf{k}_0, _)).\mathsf{new}\ \tilde{r}.\mathsf{out}(c_a, \mathsf{senc}(\mathsf{senc}(x, a, r_2), \mathsf{k}_0, r_1)) & (1) \\
| & !\,\mathsf{in}(c_i, \mathsf{senc}(x.u_i, \mathsf{k}_0, _)).\mathsf{new}\ r.\mathsf{out}(c_i, \mathsf{senc}(x, \mathsf{k}_1, r)) & (2) \\
| & !\,\mathsf{in}(c_i, \mathsf{senc}(x.u_i, \mathsf{k}_1, _)).\mathsf{new}\ r.\mathsf{out}(c_i, \mathsf{senc}(x, \mathsf{k}_1, r)) & (3) \\
| & !\,\mathsf{in}(c', \mathsf{senc}(\epsilon, \mathsf{k}_1, _)).\mathsf{new}\ r.\mathsf{out}(c', \mathsf{senc}(\epsilon, \mathsf{k}_2, r)) & (4)
\end{array}$$

where i ranges in $\{1,\dots,n\}$ and a in A.

$$
\begin{array}{lll}
P_V := & !\,\mathsf{in}(c,\mathsf{start}).\mathsf{new}\ r.\mathsf{out}(c,\mathsf{senc}(\epsilon,\mathsf{k}'_0,r)) & (\mathsf{start})\\
| & !\,\mathsf{in}(c_a,\mathsf{senc}(x,\mathsf{k}'_0,\_)).\mathsf{new}\ \tilde{r}.\mathsf{out}(c_a,\mathsf{senc}(\mathsf{senc}(x,a,r_2),\mathsf{k}'_0,r_1)) & (1)\\
| & !\,\mathsf{in}(c_i,\mathsf{senc}(x.v_i,\mathsf{k}'_0,\_)).\mathsf{new}\ r.\mathsf{out}(c_i,\mathsf{senc}(x,\mathsf{k}'_1,r)) & (2')\\
| & !\,\mathsf{in}(c_i,\mathsf{senc}(x.v_i,\mathsf{k}'_1,\_)).\mathsf{new}\ r.\mathsf{out}(c_i,\mathsf{senc}(x,\mathsf{k}'_1,r)) & (3')\\
| & !\,\mathsf{in}(c_i,\mathsf{senc}(x.w,\mathsf{k}'_0,\_)).\mathsf{new}\ \tilde{r}.\mathsf{out}(c_i,\mathsf{senc}(x.w,\mathsf{k}'_3,r_1)) & (2'a)\\
| & !\,\mathsf{in}(c_i,\mathsf{senc}(x.w,\mathsf{k}'_1,\_)).\mathsf{new}\ \tilde{r}.\mathsf{out}(c_i,\mathsf{senc}(x.w,\mathsf{k}'_3,r_1)) & (3'a)\\
| & !\,\mathsf{in}(c_i,\mathsf{senc}(w',\mathsf{k}'_0,\_)).\mathsf{new}\ r.\mathsf{out}(c_i,\mathsf{senc}(w',\mathsf{k}'_3,r)) & (2'b)\\
| & !\,\mathsf{in}(c_i,\mathsf{senc}(w',\mathsf{k}'_1,\_)).\mathsf{new}\ r.\mathsf{out}(c_i,\mathsf{senc}(w',\mathsf{k}'_3,r)) & (3'b)\\
| & !\,\mathsf{in}(c_i,\mathsf{senc}(x,\mathsf{k}'_3,\_)).\mathsf{new}\ r.\mathsf{out}(c_i,\mathsf{senc}(x,\mathsf{k}'_3,r)) & (5')\\
| & !\,\mathsf{in}(c',\mathsf{senc}(\epsilon,\mathsf{k}'_1,\_)).\mathsf{out}(c',\mathsf{start}) & (4')\\
| & !\,\mathsf{in}(c',\mathsf{senc}(x.\alpha,\mathsf{k}'_1,\_)).\mathsf{new}\ \tilde{r}.\mathsf{out}(c',\mathsf{senc}(x.\alpha,\mathsf{k}'_2,r_1)) & (4'a)\\
| & !\,\mathsf{in}(c',\mathsf{senc}(x,\mathsf{k}'_3,\_)).\mathsf{new}\ r.\mathsf{out}(c',\mathsf{senc}(x,\mathsf{k}'_2,r)) & (4'b)
\end{array}
$$

where i ranges in $\{1,\dots,n\}$, a and α in A, and for each $i \in \{1,\dots,n\}$, w in W_i and w' in W'_i.

Lemma 1. *We have that $P_U \sqsubseteq P_V$ if, and only if, PCP has no solution.*

Proof. We prove successively the two implications.

($\Rightarrow$) *If PCP has a solution then $P_U \not\sqsubseteq P_V$.* If PCP has a solution, there exists a word $u = \alpha_1 \dots \alpha_m \in A^+$, $p \in \mathbb{N}$ and $(i_k)_{1\le k\le p} \in \mathbb{N}^p$ such that

$$u = u_{i_1} \cdot \ldots \cdot u_{i_p} = v_{i_1} \cdot \ldots \cdot v_{i_p}.$$

From this word and sequence, the attacker can build the term u^* representing the word $\alpha_1 \dots \alpha_m$ from the letters $\alpha_i \in A$ with branches (1) and then output one by one the tiles u_{i_p} to u_{i_1} using (2) and (3). Let tr be the trace of the protocol P_U following the sequence $i_1, \dots, i_p$:

$$
\begin{array}{l}
\mathsf{tr} \stackrel{\mathsf{def}}{=} \mathsf{in}(c,\mathsf{start}).\mathsf{out}(c,\mathsf{w}_1).\mathsf{in}(c_{\alpha_1},\mathsf{w}_1).\mathsf{out}(c_{\alpha_1},\mathsf{w}_2)\dots\mathsf{in}(c_{\alpha_m},\mathsf{w}_m).\mathsf{out}(c_{\alpha_m},\mathsf{w}_{m+1}).\\
\quad \mathsf{in}(c_{i_p},\mathsf{w}_{m+1}).\mathsf{out}(c_{i_p},\mathsf{w}_{m+2})\dots\mathsf{in}(c_{i_1},\mathsf{w}_{p+m}).\mathsf{out}(c_{i_1},\mathsf{w}_{p+m+1}).\\
\quad \mathsf{in}(c',\mathsf{w}_{m+p+1}).\mathsf{out}(c',\mathsf{w}_{m+p+2})
\end{array}
$$

The trance tr models the fact that, given u^*, P_U can remove one by one the tiles u_{i_p} to u_{i_1} to reach the empty word and hence output the message $\mathsf{senc}(\epsilon,\mathsf{k}_2,r)$. In this execution, except for the first input on channel c, the only input recipes are variables of the frame w_k and no equality holds in the resulting frame ϕ, as the attacker ignores the keys that are used to encrypt, and all outputted message use different random seeds; thus all messages look fresh.

We claim that there exists no equivalent trace in P_V. Indeed, as the pattern matching operated by process parts $(2')$, $(3')$, $(2'a)$, $(3'a)$ and $(2'b)$, $(3'b)$ is exclusive, given u^* as an input (which is necessary as the channels used to build it are known to the attacker), P_V has no choice but to remove tiles $v_{i_p} \in V_{i_p}$ to $v_{i_1} \in V_{i_1}$ and output start on channel c' as u is a Post word. Any other trace would either lead to a mismatch on the channels or an improper filtering in P_V. Then the test $\mathsf{w}_{m+p+2} = \mathsf{start}$ is evaluated to true in ψ (the frame resulting from this execution in P_V) but false in ϕ. So (tr,ϕ) has no equivalent trace in P_V, i.e. $P_U \not\sqsubseteq P_V$.

($\Leftarrow$) *If PCP has no solution then $P_U \sqsubseteq P_V$.* First, note that, the only way for the attacker to build a trace is to start with branch (start) and use an arbitrary

number of times the branches (1) before having to be able to fire any subsequent one. For this reason, any trace (tr, ϕ) of P_U can be associated to a set of words $u^* \in A^*$, obtained from the different uses of branches (1). If $u^* = \epsilon$, because transition (4) can only be fired with a ciphertext with key k_1 and the tiles of U are non-empty, then the trace actually consists on branch (start) only, which has an obvious equivalent in P_V. Given a trace $(\mathsf{tr}, \phi) \in \mathsf{trace}(P_U)$, we aim at showing there exists an equivalent trace $(\mathsf{tr}, \psi) \in \mathsf{trace}(P_V)$. Two cases can occur for any trace $(\mathsf{tr}, \phi) \in \mathsf{trace}(P_U)$:

- tr contains no input on channel c': then ϕ contains no equalities, due to fresh nonces for every output and ignorance of the keys. (tr, ψ) can be built by following the sequence of channels used in tr and choosing the adequate filtering ($(2')$, $(3')$, $(2'a)$, $(3'a)$, $(2'b)$ or $(3'c)$). It is always possible to do so, as the definition of sets W_i and W'_i ensure that every term built by the attacker can be handled on any channel c_i. As for ϕ, the frame ψ will not contain any equality, hence demonstrating the equivalence of (tr, ϕ) and (tr, ψ).

- tr contains an input on channel c': then a word u^* associated to tr, as mentioned before, is made of tiles of U_i. Indeed, the only way to activate an input on c' is to go through the branches (2) and (3) by unstacking the said tiles. Then, because PCP has no solution, such a word u^* cannot be a Post word and thus u^* cannot be decomposed in tiles of V_i with the same sequence of indices: because the filtering in P_V is also exhaustive, messages outputted by P_V (up to replays) from a certain point will be encrypted by k'_3, which will enable P_V, through the part $(5')$, to match inputs and outputs on any channel c_i. Finally, parts $(4'a)$ and $(4'b)$ allow P_V to match the outputs of P_U on c'. For the same reason as before, ϕ and ψ are statically equivalent as the messages appears to be random to the attacker and thus contain no equalities.

Hence, for all traces of P_U there exists an equivalent trace of P_V, i.e. $P_U \sqsubseteq P_V$. □

Theorem 2 directly follows from Lemma 1 and the undecidability of the Post Correspondence Problem.

B Getting rid of the attacker

This section is devoted to the proof of Proposition 1 introduced in Section 4.2.

Proposition 1. *Let P and Q be two protocols in $\mathcal{C}_{\mathsf{pp}}$. We have that:*

$$P \approx Q \text{ if, and only if, } P' \approx_{\mathsf{fwd}} Q' \text{ for some } (P', Q') \in \mathcal{T}_{\mathsf{fwd}}(P, Q).$$

The general proof is articulated as follows: we first show in Lemma 3 that, for protocols in $\mathcal{C}_{\mathsf{pp}}$, our semantics can be restricted slightly by enforcing outputs right after their corresponding inputs without altering our definition of trace equivalence. Then, in Lemma 5, we show that deducible keys can actually be given *a priori* to the attacker for such protocols, enabling us in Lemma 6 to prove that trace equivalence with respect to a forwarder for protocols whose deducible keys are revealed is itself equivalent to regular trace equivalence in our setting. Proposition 1 finally wraps those lemmas together.

Lemma 2. *Let P and Q be two protocols in $\mathcal{C}_{\mathsf{pp}}$. We have that:*

$$P \approx Q \text{ if, and only if, } P \mid !\mathsf{in}(c,x).\mathsf{out}(c,x) \approx Q \mid !\mathsf{in}(c,x).\mathsf{out}(c,x)$$

where c is a channel name that does not occur in P and Q.

Proof. Obvious. It just enables the attacker to store computations inside the frame.

The *height* of a recipe is defined as expected, to be 0 for constants and variables and to be 1 plus the maximum height of its arguments for a term with a function as a top-level symbol.

Lemma 3 (equivalence of semantics)**.** *Let P and Q be two protocols in $\mathcal{C}_{\mathsf{pp}}$. We have that $P \approx Q$ in the usual semantics described in Section 2.2 if, and only if, $P|!\mathsf{in}(c,x).\mathsf{out}(c,x) \approx Q|!\mathsf{in}(c,x).\mathsf{out}(c,x)$ in the semantics where c is a channel name that does not occur in P and Q, and:*

1. *every trace is made of segments $\mathsf{in}(c_i,R).\mathsf{out}(c_i,\mathsf{w})$ for various channel names c_i and variables of the frame w,*
2. *recipes in every input of the trace on a channel different from c are variables of the frame or the constant start,*
3. *recipes in every input on channel c are of maximal height 1,*
4. *recipes in the definition of static equivalence are variables of the frames only or the constant start.*

Proof. Let (tr,ϕ) be a trace of P, we define inductively $(\bar{\mathsf{tr}},\bar{\phi}) \in \mathsf{trace}(P)$ to be a new trace satisfying the conditons of Lemma 3:

- if tr does not contain any output: then $(\bar{\mathsf{tr}},\bar{\phi})$ is the empty trace with the empty frame,
- if $\mathsf{tr} = \mathsf{in}(c_i,R).\mathsf{tr}_1.\mathsf{out}(c_i,\mathsf{w}).\mathsf{tr}_2$, where $\mathsf{in}(c_i,R)$ is the first input of tr and $\mathsf{out}(c_i,\mathsf{w})$ its corresponding output in tr. If R is of height M, because encryption and decryption have to be made with atomic keys only, R is composed of $M-1$ encryption or decryption symbols. If we call R_k the term at depth $M-k$, $R_k = \mathsf{senc}(u,v,w)$ or $R_k = \mathsf{sdec}(u,v)$ where u is a term and v,w are constants, names or variable of $dom(\phi)$. Then:

  $$\bar{\mathsf{tr}} = \mathsf{in}(c,R'_1).\mathsf{out}(c,\mathsf{w}_1)\ldots\mathsf{in}(c,R'_M).\mathsf{out}(c,\mathsf{w}_M).\mathsf{in}(c_i,\mathsf{w}_M).\mathsf{out}(c_i,\mathsf{w}).\overline{\mathsf{tr}_1.\mathsf{tr}_2}$$

  where $\mathsf{w}_1,\ldots,\mathsf{w}_M$ are new variables in the frame, $R'_1 = R_1$ and $R'_k = R_k\{u \mapsto \mathsf{w}_{k-1}\}$ for $k \in \{2,\ldots,M\}$. Note that variables in input may have to be reindexed to fit with our semantics. If $(\mathsf{tr},\phi) \in \mathsf{trace}(P)$ then $(\bar{\mathsf{tr}},\bar{\phi}) \in \mathsf{trace}(P)$. Moreover, terms of ϕ are present in $\bar{\phi}$.

Moreover notice that this transformation is deterministic: to a sequence of labels tr we can associate a unique $\bar{\mathsf{tr}}$; and $\bar{\mathsf{tr}}$ satisfies all of the needed properties. If $P \approx Q$, then we directly get that $P|!\mathsf{in}(c,x).\mathsf{out}(c,x) \approx Q|!\mathsf{in}(c,x).\mathsf{out}(c,x)$ thanks to Lemma 2 and because the constraints in the new semantics actually reduces symmetrically the set of traces of P and Q. What remains to show is that if $P \not\approx Q$, then $P|!\mathsf{in}(c,x).\mathsf{out}(c,x) \not\approx Q|!\mathsf{in}(c,x).\mathsf{out}(c,x)$ in the new semantics. For instance, let (tr,ϕ) be a trace of P. Two cases can occur:

- either there exists no frame ψ such that $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$. Indeed, in such a case $(\bar{\mathsf{tr}}, \bar{\phi}) \in \mathsf{trace}(P)$ but there exists no frame $\bar{\psi}$ such that $(\bar{\mathsf{tr}}, \bar{\psi}) \in \mathsf{trace}(Q)$. Indeed, if we consider the first input of tr which fails in Q, the corresponding input of $\bar{\mathsf{tr}}$ will fail too as its frame contains the same terms as the frame generated so far in Q.

- or there exists a frame ψ such that $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$, two recipes R_1, R_2 such that, *e.g.* $R_1 = R_2$ holds in ϕ but not in ψ.

  Similarly to what has been done to define R'_1 to R'_M from R in the definition of $\bar{\mathsf{tr}}$, we can extend $\bar{\mathsf{tr}}$ once more. We decompose R_1 (resp. R_2) into a sequence of recipes R'_1 to R'_M (resp. R''_1 to R''_N) and define tr^* as follows:

$$\mathsf{tr}^* = \bar{\mathsf{tr}}\cdot \quad \begin{array}{l} \mathsf{in}(c, R'_1).\mathsf{out}(c, \mathsf{w}_{|\bar{\phi}|+1}) \dots \mathsf{in}(c, R'_M).\mathsf{out}(c, \mathsf{w}_{|\bar{\phi}|+M}). \\ \mathsf{in}(c, R''_1).\mathsf{out}(c, \mathsf{w}_{|\bar{\phi}|+M+1}) \dots \mathsf{in}(c, R''_N).\mathsf{out}(c, \mathsf{w}_{|\bar{\phi}|+M+N}) \end{array}$$

  and $\phi^* = \bar{\phi} \cup \{\mathsf{w}_{|\bar{\phi}|+1} \triangleright u_1 \dots \mathsf{w}_{|\bar{\phi}|+M+N} \triangleright u_{M+N}\}$; and $(\mathsf{tr}^*, \psi^*) \in \mathsf{trace}(Q)$ is defined symmetrically. Because terms in ϕ (resp. in ψ) are present in $\bar{\phi}$ and ϕ^* (resp. $\bar{\psi}$ and ψ^*), the equality will be true un $\bar{\phi}$ and false in $\bar{\psi}$ if, and only if, $\mathsf{w}_{|\bar{\phi}|+M} = \mathsf{w}_{|\bar{\phi}|+M+N}$ is true in ϕ^* but false in ψ^*.

Hence equivalence is preserved with our new semantics. □

From now on, we will always consider the alternative semantics defined in Lemma 3, and every considered trace will meet the specified requirements. In particular, we always assume protocols contain the branch $!\mathsf{in}(c, x).\mathsf{out}(c, x)$.

Definition 5. *Let P be a protocol in $\mathcal{C}_{\mathsf{pp}}$ (with a branch $!\mathsf{in}(c, x).\mathsf{out}(c, x)$). A term t is* deducible *in P if there exists a trace (tr, ϕ) of P and $\mathsf{w} \in dom(\phi)$ such that $\mathsf{w}\phi = t$.*

Lemma 4. *Let P and Q be two protocols in $\mathcal{C}_{\mathsf{pp}}$, $\mathcal{K}_P$ (resp. $\mathcal{K}_Q$) be the set of deducible constants of sort key that occur in P (resp. Q), if $P \approx Q$ then there exists a unique bijection α from $\mathcal{K}_P$ to $\mathcal{K}_Q$ such that for every trace $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ there exists a trace $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$ such that for any recipe R and any $\mathsf{k} \in \mathcal{K}_P$:*

- *$R\phi\downarrow$ is of sort s if, and only if, $R\psi\downarrow$ is of sort s; where $\mathsf{s} \in \{\mathsf{SimKey}, \mathsf{PubKey}, \mathsf{PrivKey}\}$.*
- *$R\phi\downarrow = \mathsf{k}$ if, and only if, $R\psi\downarrow = \alpha(\mathsf{k})$; and symmetrically for Q.*
- *$R\phi\downarrow = \mathsf{k}^{-1}$ if, and only if, $R\psi\downarrow = (\alpha(\mathsf{k}))^{-1}$.*

Proof. We can describe α as a relation in the following way: for every $\mathsf{k} \in \mathcal{K}_P$ of sort s, and every trace $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ and every $\mathsf{w} \in dom(\phi)$ such that $\mathsf{w}\phi = \mathsf{k}$, we define $\alpha(\mathsf{k}) = \mathsf{w}\psi$ where ψ is the only frame such that $(\mathsf{tr}, \psi) \in Q$. The existence of such a frame comes from the trace equivalence of P and Q, whereas unicity is a consequence of the determinism of protocols in $\mathcal{C}_{\mathsf{pp}}$.

We now need to prove that our definition of α is sound and unambiguous. To do so, we show that:

- $\mathsf{w}\psi$ is a constant of sort s. We have that there exists a trace $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ such that $w\phi = \mathsf{k} \in K_P$. Since $P \approx Q$, consider the corresponding equivalence trace $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$. By definition of static equivalence, we necessarily have that $\mathsf{w}\psi$ is a constant of sort s. Otherwise, we

would have that $\mathsf{senc}(\mathsf{start}, \mathsf{w}, r_i)\phi \in \mathcal{T}(\Sigma, \mathcal{N})$ whereas $\mathsf{senc}(\mathsf{start}, \mathsf{w}, r_i)\psi \notin \mathcal{T}(\Sigma, \mathcal{N})$ if $\mathsf{s} = \mathsf{SimKey}$(it is not properly sorted). The same argument applies with aenc and sign for s equal to PubKey and PrivKey respectively.

- $|\mathcal{K}_P| = |\mathcal{K}_Q|$. Suppose *ad absurdum* that, for instance, $|\mathcal{K}_P| < |\mathcal{K}_Q|$. Because every element of $\mathcal{K}_Q$ is deducible, there exists $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$ such that for all $\mathsf{k} \in \mathcal{K}_Q$, there exists $\mathsf{w}_\mathsf{k} \in dom(\psi)$ such that $\mathsf{w}_\mathsf{k}\psi = \mathsf{k}$. In particular, if $k \neq k'$, $\mathsf{w}_k\psi \neq \mathsf{w}_{k'}\psi$. Because $P \approx Q$, there exists a frame ϕ such that $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$. As $|\mathcal{K}_P| < |\mathcal{K}_Q|$, there exist two distinct keys k and k' such that $\mathsf{w}_\mathsf{k}\phi = \mathsf{w}_{\mathsf{k}'}\phi$, keys in ψ have to be matched by keys in ϕ according to the previous item. Hence ϕ and ψ are not statically equivalent, contradicting the trace equivalence of P and Q.

- α is indeed a function. Suppose there exist a trace $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$, $\mathsf{w}_i \in dom(\phi)$ and a corresponding equivalence trace $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$ such that $\mathsf{w}_i\phi = \mathsf{k}$ and $\mathsf{w}_i\psi = \mathsf{k}'$; a trace $(\mathsf{tr}', \phi') \in \mathsf{trace}(P)$, $\mathsf{w}_j \in dom(\phi')$ and a corresponding equivalence trace $(\mathsf{tr}', \psi') \in \mathsf{trace}(Q)$ such that $\mathsf{w}_j\phi' = \mathsf{k}$ but $\mathsf{w}_j\psi' = \mathsf{k}''$ with $\mathsf{k}' \neq \mathsf{k}''$. Consider the trace $\mathsf{tr} \cdot \mathsf{tr}'$. Up to a reindexing of the variables and renaming of some names in ϕ' and ψ', $(\mathsf{tr} \cdot \mathsf{tr}', \phi \cup \phi') \in \mathsf{trace}(P)$ and $(\mathsf{tr} \cdot \mathsf{tr}', \psi \cup \psi') \in \mathsf{trace}(P)$. Then test $\mathsf{w}_i = \mathsf{w}_{|\phi|+j}$ would be true in $\phi \cup \phi'$ but false in $\psi \cup \psi'$ as soon as $\mathsf{k}' \neq \mathsf{k}''$.

Then we show that α is an injection: given k, k' two distinct elements of $\mathcal{K}_P$, $\alpha(\mathsf{k}) \neq \alpha(\mathsf{k}')$: suppose, as previously, there exist a trace $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$, $\mathsf{w}_i \in dom(\phi)$ and a corresponding equivalence trace $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$ such that $\mathsf{w}_i\phi = \mathsf{k}$ and $\mathsf{w}_i\psi = \alpha(\mathsf{k})$; a trace $(\mathsf{tr}', \phi') \in \mathsf{trace}(P)$, $\mathsf{w}_j \in dom(\phi')$ and a corresponding equivalence trace $(\mathsf{tr}', \psi') \in \mathsf{trace}(Q)$ such that $\mathsf{w}_j\phi' = \mathsf{k}'$ but $\mathsf{w}_j\psi' = \alpha(\mathsf{k})$ with $\mathsf{k} \neq \mathsf{k}'$. Consider the trace $\mathsf{tr}.\mathsf{tr}'$. Up to a reindexing of the variables in ϕ' and ψ', $(\mathsf{tr}.\mathsf{tr}', \phi \cup \phi') \in \mathsf{trace}(P)$ and $(\mathsf{tr}.\mathsf{tr}', \psi \cup \psi') \in \mathsf{trace}(P)$. Then test $\mathsf{w}_i = \mathsf{w}_{|\phi|+j}$ would be true in $\phi \cup \phi'$ but false in $\psi \cup \psi'$.

We prove now that α satisfies the second condition on recipes. If $R\phi\!\downarrow$ is a message, then so is $R\psi\!\downarrow$, by definition of static equivalence. Then the same arguments as those developped in the definition of α can be extended to arbitrary recipes, hence proving the existence of α.

To show that α satisfies the last condition on recipes, suppose that $\mathsf{k} \in \mathcal{K}_P$,$R\phi\!\downarrow = \mathsf{k}^{-1}$. As previously shown, $R\psi\!\downarrow = \alpha(\mathsf{k}^{-1})$. We want to prove that $\alpha(\mathsf{k}^{-1}) = (\alpha(\mathsf{k}))^{-1}$. If k is of sort SimKey, the result is obvious as $\mathsf{k}^{-1} = \mathsf{k}$ for any such key. Suppose k is of sort PubKey. We have now that there exists a trace $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ such that $w\phi = \mathsf{k} \in K_P$. Since $P \approx Q$, consider the corresponding equivalence trace $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$. Consider the recipes $R_1 = \mathsf{aenc}(\mathsf{start}, \mathsf{w}, n)$ and $R_2 = \mathsf{adec}(R_1, R)$. Then $R_2\phi\!\downarrow = \mathsf{start}$ and $R_2\psi\!\downarrow = \mathsf{start}$ if, and only if, $R\psi\!\downarrow = (\mathsf{w}\psi)^{-1}$. As we have already proved that α preserves sorts, we get that $R_2\psi\!\downarrow$ is of sort msg if, and only if, $\alpha(\mathsf{k}^{-1}) = R\psi\!\downarrow = \mathsf{w}\psi = (\alpha(\mathsf{k}))^{-1}$. Hence α is compatible with the inverse function. The same argument can be used if k is of sort PrivKey with sign and check.

Finally we prove the unicity of such a bijection: suppose there were α' an adequate bijection and $\mathsf{k} \in \mathcal{K}_P$ such that $\alpha(\mathsf{k}) \neq \alpha'(\mathsf{k})$. By definition of α, for every trace $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ and every $\mathsf{w} \in dom(\phi)$ such that $\mathsf{w}\phi = \mathsf{k}$, $\alpha(\mathsf{k}) = \mathsf{w}\psi$. But as α' satisfy a similar properties on recipes, with $R = \mathsf{w}$, we get that $\mathsf{w}\psi = \alpha'(\mathsf{k})$, contradicting our hypothesis. Hence α is unique. □

Definition 6 (disclosing keys)**.** *Let P be a protocol in $\mathcal{C}_{\mathsf{pp}}$, K be a set of constants of sort* key *that occur in P. If for every $\mathsf{k} \in K$ there exist a channel name c_{k} and a branch $!\mathsf{in}(c_{\mathsf{k}}, \mathsf{start}).\mathsf{out}(c_{\mathsf{k}}, \mathsf{k})$ in P, then P is said* to disclose K.

Lemma 5. *Let P and Q be two protocols in $\mathcal{C}_{\mathsf{pp}}$, S (resp. S') the set of key of P (resp. Q), $P \approx Q$ if, and only if, there exist two sets $K \subseteq S$ and $K' \subseteq S'$ and a bijection $\alpha : K \to K'$ such that $\bar{P} \approx \bar{Q}$ where:*

$$\begin{array}{rcl}
\bar{P} = P & | & !\mathsf{in}(c^0, \mathsf{start}).\mathsf{out}(c^0, 0) | !\mathsf{in}(c^1, \mathsf{start}).\mathsf{out}(c^1, 1) \\
& | & \left| \limits_{\mathsf{k} \in K} !\mathsf{in}(c_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{start}).\mathsf{out}(c_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{k}) \right| \left| \limits_{\mathsf{k} \in S \setminus K} !\mathsf{in}(c, \mathsf{k}).\mathsf{out}(c, 0) \right. \\
\bar{Q} = Q & | & !\mathsf{in}(c^0, \mathsf{start}).\mathsf{out}(c^0, 0) | !\mathsf{in}(c^1, \mathsf{start}).\mathsf{out}(c^1, 1) \\
& | & \left| \limits_{\mathsf{k} \in K} !\mathsf{in}(c_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{start}).\mathsf{out}(c_{\mathsf{k},\alpha(\mathsf{k})}, \alpha(\mathsf{k})) \right| \left| \limits_{\mathsf{k} \in S' \setminus K'} !\mathsf{in}(c, \mathsf{k}).\mathsf{out}(c, 1) \right.
\end{array}$$

and 0, 1 are new constants, c^0, c^1, the $c_{\mathsf{k},\alpha(\mathsf{k})}$ and c are fresh channels. Moreover, assuming the existence of such sets and bijection such that $\bar{P} \approx \bar{Q}$, the two protocols are disclosing their deducible keys. We call $\mathcal{T}^1(P, Q)$ the set of such pairs $(\bar{P}, \bar{Q})$ of modified protocols.

Proof. If $P \approx Q$, by Lemma 4, for $K = \mathcal{K}_P$ and $K' = \mathcal{K}_Q$, we get the existence of such a bijection α. Because keys in $S \setminus \mathcal{K}_P$ and $S' \setminus \mathcal{K}_Q$ are not deducible, the branches on channel c can never be triggered. Conversely, as $P \approx Q$, any trace of P (resp. Q) inputting or outputting on a channel $c_{\mathsf{k},\alpha(\mathsf{k})}$ for k in $\mathcal{K}_P$ can be matched in Q (resp. P) as any reduction to a key k in P corresponds to a reduction to $\alpha(\mathsf{k})$ in Q and for every couple $(\mathsf{k}, \mathsf{k}^{-1})$ of deducible keys, for any recipe reducing to k^{-1} in P, the same recipe reduces to $\alpha(\mathsf{k})^{-1}$ in Q, thanks to the properties of α described in Lemma 4.

For the converse implication, remark that necessarily $\mathcal{K}_P \subseteq K$ and $\mathcal{K}_Q \subseteq K'$: otherwise suppose there exists, for instance, $\mathsf{k} \in \mathcal{K}_P \setminus K$. Because k is deducible, there exists a trace $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ and $\mathsf{w} \in dom(\phi)$ such that $\mathsf{w}\phi = \mathsf{k}$. (tr, ϕ) is also a trace of $\bar{P}$. Consider the trace

$$\mathsf{tr}' = \mathsf{tr}.\mathsf{in}(c, \mathsf{w}).\mathsf{out}(c, \mathsf{w}_{|\phi|+1}).\mathsf{in}(c^0, \mathsf{start}).\mathsf{out}(c^0, \mathsf{w}_{|\phi|+2}).\mathsf{in}(c^1, \mathsf{start}).\mathsf{out}(c^1, \mathsf{w}_{|\phi|+3})$$

along with its frame $\phi' = \phi \cup \{\mathsf{w}_{|\phi|+1} \triangleright 0, \mathsf{w}_{|\phi|+2} \triangleright 0, \mathsf{w}_{|\phi|+3} \triangleright 1\}$. If $\bar{P} \approx \bar{Q}$, then there exists $(\mathsf{tr}', \psi') \in \mathsf{trace}(\bar{Q})$ such that ϕ and ψ are statically equivalent. But any output on c in Q leads to the constant 1, breaking static equivalence. And symmetrically for any $\mathsf{k} \in \mathcal{K}_Q \setminus K'$.

Finally we need to prove that $\bar{P} \approx \bar{Q}$ implies $P \approx Q$. For every trace $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$, $(\mathsf{tr}, \phi) \in \mathsf{trace}(\bar{P})$, and as $\bar{P} \approx \bar{Q}$, there exists a trace $(\mathsf{tr}, \psi) \in \mathsf{trace}(\bar{Q})$ such that ϕ is statically equivalent to ψ. Because c^0, c^1, c and the $c_{\mathsf{k},\alpha(\mathsf{k})}$ are new channels, tr does not use transitions on those, thus $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$. The same goes for any trace of Q, hence showing the trace equivalence of P and Q. □

We shall now use the forwarder as defined in Section 4.2.

Lemma 6 (active attacker vs. forwarder)**.** *Let P and Q be two protocols in $\mathcal{C}_{\mathsf{pp}}$ respectively disclosing two sets of keys K and K' as in Lemma 5. Then $P \approx Q$*

if, and only if, $\bar{P} \approx_{\mathsf{fwd}} \bar{Q}$ *where:*

$$
\begin{array}{rll}
\bar{P} = P & | & \displaystyle\bigsqcup_{\mathsf{k}\in K^{\mathsf{SimKey}}} !\mathsf{in}(c^{\mathsf{senc}}_{\mathsf{k},\alpha(\mathsf{k})}, x).\mathsf{new}\ n.\mathsf{out}(c_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{senc}(x,\mathsf{k},n)) \\
& | & \displaystyle\bigsqcup_{\mathsf{k}\in K^{\mathsf{SimKey}}} !\mathsf{in}(c^{\mathsf{sdec}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{senc}(x,\mathsf{k},y)).\mathsf{out}(c^{\mathsf{sdec}}_{\mathsf{k},\alpha(\mathsf{k})}, x) \\
& | & \displaystyle\bigsqcup_{\mathsf{k}\in K^{\mathsf{PubKey}}} !\mathsf{in}(c^{\mathsf{aenc}}_{\mathsf{k},\alpha(\mathsf{k})}, x).\mathsf{new}\ n.\mathsf{out}(c^{\mathsf{aenc}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{aenc}(x,\mathsf{k},n)) \\
& | & \displaystyle\bigsqcup_{\mathsf{k}\in K^{\mathsf{PrivKey}}} !\mathsf{in}(c^{\mathsf{adec}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{aenc}(x,\mathsf{k},y)).\mathsf{out}(c^{\mathsf{adec}}_{\mathsf{k},\alpha(\mathsf{k})}, x) \\
& | & \displaystyle\bigsqcup_{\mathsf{k}\in K^{\mathsf{PrivKey}}} !\mathsf{in}(c^{\mathsf{sign}}_{\mathsf{k},\alpha(\mathsf{k})}, x).\mathsf{new}\ n.\mathsf{out}(c^{\mathsf{sign}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{sign}(x,\mathsf{k},n)) \\
& | & \displaystyle\bigsqcup_{\mathsf{k}\in K^{\mathsf{PubKey}}} !\mathsf{in}(c^{\mathsf{check}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{sign}(x,\mathsf{k},y)).\mathsf{out}(c^{\mathsf{check}}_{\mathsf{k},\alpha(\mathsf{k})}, x) \\
\bar{Q} = Q & | & \displaystyle\bigsqcup_{\mathsf{k}\in K^{\mathsf{SimKey}}} !\mathsf{in}(c^{\mathsf{senc}}_{\mathsf{k},\alpha(\mathsf{k})}, x).\mathsf{new}\ n.\mathsf{out}(c^{\mathsf{senc}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{senc}(x,\alpha(\mathsf{k}),n)) \\
& | & \displaystyle\bigsqcup_{\mathsf{k}\in K^{\mathsf{SimKey}}} !\mathsf{in}(c^{\mathsf{sdec}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{senc}(x,\alpha(\mathsf{k}),y)).\mathsf{out}(c^{\mathsf{sdec}}_{\mathsf{k},\alpha(\mathsf{k})}, x) \\
& | & \displaystyle\bigsqcup_{\mathsf{k}\in K^{\mathsf{PubKey}}} !\mathsf{in}(c^{\mathsf{aenc}}_{\mathsf{k},\alpha(\mathsf{k})}, x).\mathsf{new}\ n.\mathsf{out}(c^{\mathsf{aenc}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{aenc}(x,\alpha(\mathsf{k}),n)) \\
& | & \displaystyle\bigsqcup_{\mathsf{k}\in K^{\mathsf{PrivKey}}} !\mathsf{in}(c^{\mathsf{adec}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{aenc}(x,\alpha(\mathsf{k}),y)).\mathsf{out}(c^{\mathsf{adec}}_{\mathsf{k},\alpha(\mathsf{k})}, x) \\
& | & \displaystyle\bigsqcup_{\mathsf{k}\in K^{\mathsf{PrivKey}}} !\mathsf{in}(c^{\mathsf{sign}}_{\mathsf{k},\alpha(\mathsf{k})}, x).\mathsf{new}\ n.\mathsf{out}(c^{\mathsf{sign}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{sign}(x,\alpha(\mathsf{k}),n)) \\
& | & \displaystyle\bigsqcup_{\mathsf{k}\in K^{\mathsf{PubKey}}} !\mathsf{in}(c^{\mathsf{check}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{check}(x,\alpha(\mathsf{k}),y)).\mathsf{out}(c^{\mathsf{check}}_{\mathsf{k},\alpha(\mathsf{k})}, x)
\end{array}
$$

where K^{s} *denotes the keys of sort* s *of* K*. We call* $\mathcal{T}^2$ *the transformation taking a pair of protocols* (P, Q) *satisfying the aforementioned condition and returning the pair* $(\bar{P}, \bar{Q})$ *presently defined.*

Proof. We recall, that, as a consequence of Lemma 5, we necessarily have that $\mathcal{K}_P \subseteq K$ and $\mathcal{K}_Q \subseteq K'$. Because protocols P and $\bar{P}$ (resp. Q and $\bar{Q}$) disclose all their deducible keys, their exists a trace (tr_0, ϕ_0) of P and $\bar{P}$ (resp. (tr_0, ψ_0) a trace of Q and $\bar{Q}$) defined as follows:

$$\mathsf{tr}_0 = \mathsf{in}(c_{\mathsf{k}_1,\alpha(\mathsf{k}_1)}, \mathsf{start}).\mathsf{out}(c_{\mathsf{k}_1,\alpha(\mathsf{k}_1)}, \mathsf{w}_1^0) \ldots \mathsf{in}(c_{\mathsf{k}_n,\alpha(\mathsf{k}_n)}, \mathsf{start}).\mathsf{out}(c_{\mathsf{k}_n,\alpha(\mathsf{k}_n)}, \mathsf{w}_n^0)$$

for $\mathsf{k}_1, \ldots, \mathsf{k}_n \in \mathcal{K}_P$, and $\phi_0 = \{\mathsf{w}_1^0 \triangleright \mathsf{k}_1, \ldots \mathsf{w}_n^0 \triangleright \mathsf{k}_n\}$, and symmetrically for Q and $\bar{Q}$. We can always assume a trace of P or $\bar{P}$ (resp. of Q or $\bar{Q}$) starts with the prefix tr_0 and contains the frame ϕ_0.

First, suppose $\bar{P} \not\approx_{\mathsf{fwd}} \bar{Q}$: for instance, suppose there exists $(\mathsf{tr}, \phi) \in \mathsf{trace}(\bar{P})$ such that there is no equivalent frame ψ such that $(\mathsf{tr}, \psi) \in \mathsf{trace}(\bar{Q})$. Let us define a corresponding trace $(\mathsf{tr}', \phi) \in \mathsf{trace}(P)$:

- every sequence $\mathsf{in}(c^{\mathsf{senc}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{w}).\mathsf{out}(c^{\mathsf{senc}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{w}')$ in tr is replaced by the sequence $\mathsf{in}(c, \mathsf{senc}(\mathsf{w}, \mathsf{w}_k^0, n)).\mathsf{out}(c, \mathsf{w}')$ in tr' where n is a fresh name, which is a recipe of height 1.
- every sequence $\mathsf{in}(c^{\mathsf{sdec}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{w}).\mathsf{out}(c^{\mathsf{sdec}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{w}')$ in tr is replaced by the sequence $\mathsf{in}(c, \mathsf{sdec}(\mathsf{w}, \mathsf{w}_k^0)).\mathsf{out}(c, \mathsf{w}')$ in tr' which is a recipe of height 1.
- every sequence $\mathsf{in}(c^{\mathsf{aenc}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{w}).\mathsf{out}(c^{\mathsf{aenc}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{w}')$ in tr is replaced by the sequence $\mathsf{in}(c, \mathsf{aenc}(\mathsf{w}, \mathsf{w}_k^0, n)).\mathsf{out}(c, \mathsf{w}')$ in tr' where n is a fresh name, which is a recipe of height 1.

- every sequence $\mathsf{in}(c^{\mathsf{adec}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{w}).\mathsf{out}(c^{\mathsf{adec}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{w}')$ in tr is replaced by the sequence $\mathsf{in}(c, \mathsf{adec}(\mathsf{w}, \mathsf{w}^0_k)).\mathsf{out}(c, \mathsf{w}')$ in tr' which is a recipe of height 1.

- every sequence $\mathsf{in}(c^{\mathsf{sign}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{w}).\mathsf{out}(c^{\mathsf{sign}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{w}')$ in tr is replaced by the sequence $\mathsf{in}(c, \mathsf{sign}(\mathsf{w}, \mathsf{w}^0_k, n)).\mathsf{out}(c, \mathsf{w}')$ in tr' where n is a fresh name, which is a recipe of height 1.

- every sequence $\mathsf{in}(c^{\mathsf{check}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{w}).\mathsf{out}(c^{\mathsf{check}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{w}')$ in tr is replaced by the sequence $\mathsf{in}(c, \mathsf{check}(\mathsf{w}, \mathsf{w}^0_k)).\mathsf{out}(c, \mathsf{w}')$ in tr' which is a recipe of height 1.

We claim that there exists no frame ψ such that $(\mathsf{tr}', \psi) \in \mathsf{trace}(Q)$ with $\phi \sim \psi$. Indeed, because the frame are left unchanged, the input recipes match the same input patterns, and recipes holding true and false keep their truth values. So if such a frame ψ existed, (tr, ψ) would belong to $\mathsf{trace}(\bar{Q})$ and be equivalent to (tr, ϕ).

Now, suppose $P \not\approx Q$: consider the shortest trace $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$, in terms of number of transitions, such that there is no equivalent frame ψ satisfying $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$.

Through recipes of the form $\mathsf{senc}(u, v, w)$ on channel c, the attacker has the ability to use the same random seed more that once. Let us first show that we can always assume tr uses nonces at most once. If it is not the case, we build a new trace $(\tilde{\mathsf{tr}}, \tilde{\phi})$, such that ϕ is statically equivalent to $\tilde{\phi}$ for which it is the case. Suppose a random seed r is used in two recipes, two cases can occur:

- r is used with two different keys, two different plaintexts or two different function symbols: because random seeds are not filtered in our semantics, the sequence of labels where an occurence of r is replaced by a distinct r' is still a trace of P. And as equality between ciphertexts implies equality between their keys, between their plaintexts and between their function symbols, changing r into r' does not prevent any existing equality, nor introduce any. Thus we can replace r in $\tilde{\mathsf{tr}}$ by a new r'. We claim that, if there existed ψ such that $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$, then either the keys, the plaintexts or the function symbols in the two recipes are different in ψ. Indeed, if it were not the case, consider the shorter trace (tr', ϕ') obtained by deleting the transitions with the second occurence of the recipe and any later transitions. Similarly (tr', ψ') would be a shorter trace of Q. Because terms in ϕ' (resp. ψ') were already in ϕ (resp. ψ), the equality between the keys, the plaintexts or the function symbols used for the recipes would hold in ψ' but not in ϕ', thus exhibiting a witness of non-equivalence shorter than the minimal one.

- r is used with the same key, plaintext and function symbol: as the terms are identical, we can replace the later use of the recipe $\mathsf{senc}(u, v, w)$ by the variable $\mathsf{w} \in dom(\phi)$ corresponding to the output of the first occurence of this term in $\tilde{\mathsf{tr}}$. The frame of the new trace being the same as the initial one, the same equalities hold. We claim that, if there exists ψ such that $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$, then the terms used as keys, plaintexts and function symbols for these recipes are identical in ψ too. As in the previous item, if it were not the case, we could build a shorter witness of non-equivalence between P and Q, contradicting the minimality assumption.

In particular, we showed that modifiying tr into $\tilde{\mathsf{tr}}$ is a symmetric operation which preserves equalities in the two protocols: identical plaintexts and keys in (tr, ϕ) correspond to identical plaintexts and keys in (tr, ψ), whereas adding fresh nonces does not create any equality in $\tilde{\phi}$ or $\tilde{\psi}$. If (tr, ϕ) does not have any equivalent trace in Q, neither has $(\tilde{\mathsf{tr}}, \tilde{\phi})$. If there exists no frame ψ such that $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$, then there will exist no frame $\tilde{\psi}$ such that $(\tilde{\mathsf{tr}}, \tilde{\psi}) \in \mathsf{trace}(Q)$ as input filtering is not affected by our transformation. Else, if there exists ψ such that $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$ but ϕ and ψ are not statically equivalent, because our transformation preserves the terms in the frame, any pair of recipes which distinguishes between the two of them, will distinguish $\tilde{\phi}$ and $\tilde{\psi}$. So we can always assume than the random seeds occuring in the recipes $\mathsf{senc}(u, v, w)$ in (tr, ϕ) are distinct.

Let us now define a corresponding trace $(\bar{\mathsf{tr}}, \phi) \in \mathsf{trace}(\bar{P})$. According to Lemma 3, we may only consider the following pairs of input and outputs:

- the sequences $\mathsf{in}(c_i, \mathsf{w}).\mathsf{out}(c_i, \mathsf{w}')$, where $c_i \neq c$, are left unchanged
- whereas each sequence $\mathsf{in}(c, \mathsf{f}(\mathsf{w}, x, n)).\mathsf{out}(c, \mathsf{w}')$, where $x = \mathsf{k}$ or $x\phi = \mathsf{k}$ and $\mathsf{f} \in \{\mathsf{senc}, \mathsf{aenc}, \mathsf{sign}\}$, is replaced by $\mathsf{in}(c^{\mathsf{f}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{w}).\mathsf{out}(c^{\mathsf{f}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{w}')$;
- and each sequence $\mathsf{in}(c, \mathsf{g}(\mathsf{w}, x)).\mathsf{out}(c, \mathsf{w}')$, where $x = \mathsf{k}$ or $x\phi = \mathsf{k}$ and $\mathsf{g} \in \{\mathsf{sdec}, \mathsf{adec}, \mathsf{check}\}$, is replaced by $\mathsf{in}(c^{\mathsf{g}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{w}).\mathsf{out}(c^{\mathsf{sdec}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{w}')$.

The corresponding frame $\bar{\phi}$ is then defined according to our semantics.

Finally, because $(\tilde{\mathsf{tr}}, \tilde{\phi}) \in \mathsf{trace}(P)$ has no equivalent in Q, and the definition of $(\bar{\mathsf{tr}}, \bar{\phi})$ does not alter the filtering on inputs nor equalities between terms in the frame, $(\bar{\mathsf{tr}}, \bar{\phi})$ has no equivalent in $\mathsf{trace}(\bar{Q})$.

Moreover, note that for every key in K (resp. in K'), two branches of fixed size are added to $\bar{P}$ (resp. $\bar{Q}$). $\mathcal{T}_2$ can thus be computed in polynomial time, with respect to the size of its inputs.

□

Lemmas 5 and 6 enable us to define the transformation $\mathcal{T}_{\mathsf{fwd}}$ introduced in Section 4.2 as follows. For every pair of protocols (P, Q) in $\mathcal{C}_{\mathsf{pp}}$:

$$\mathcal{T}_{\mathsf{fwd}}(P, Q) = \{\mathcal{T}^2(P', Q') | (P', Q') \in \mathcal{T}^1(P, Q)\}$$

Combination of the two previous results leads to the desired result:

Proposition 1. *Let P and Q be two protocols in $\mathcal{C}_{\mathsf{pp}}$. We have that:*

$$P \approx Q \text{ if, and only if, } P' \approx_{\mathsf{fwd}} Q' \text{ for some } (P', Q') \in \mathcal{T}_{\mathsf{fwd}}(P, Q).$$

C Encoding a protocol into a real-time GPDA

This section is devoted to the proof of Theorem 3 introduced in Section 4.3.

Theorem 3. *Let P and Q in $\mathcal{C}_{\mathsf{pp}}$, we have that: $P \approx_{\mathsf{fwd}} Q \iff \mathcal{L}(\mathcal{A}_P) = \mathcal{L}(\mathcal{A}_Q)$.*

To prove this theorem, we need to present a number of intermediate results. The sketch goes as follows: we first prove in Lemma 10 that the notion of equivalence introduced in Section 4.2 can be characterised as a conjonction of

simpler properties on protocols in $\mathcal{C}_{\mathsf{pp}}$. The satisfaction of each pair of property can then encoded into real-time GPDA as described in Lemma 11. The proof of Theorem 3 resulting from the combination of the above lemmas.

Lemma 7. *Given a protocol P in $\mathcal{C}_{\mathsf{pp}}$, a trace $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ and a variable $\mathsf{w} \in dom(\phi)$, there exists a unique sequence $(c_{i_0}, \mathsf{w}_{j_0}).(c_{i_1}, \mathsf{w}_{j_1}) \dots (c_{i_p}, \mathsf{w})$ such that the sequence $\mathsf{in}(c_{i_0}, \mathsf{start}).\mathsf{out}(c_{i_0}, \mathsf{w}_{j_0})$ occurs in tr and for all $k \in \{1, \dots, p\}$, the sequence $\mathsf{in}(c_{i_k}, \mathsf{w}_{j_{k-1}}).\mathsf{out}(c_{i_k}, \mathsf{w}_{j_k})$ occurs in tr, where w is interpreted as w_{j_p}.*

Proof. Any term in ϕ is obtained by applying a branch c_{i_k} to previous element of the frame. By induction we get the desired result. □

In the following, we denote by $\mathsf{seq}_{\mathsf{tr},\phi}(\mathsf{w})$ the sequence associated to the element $\mathsf{w} \in dom(\phi)$ as defined in Lemma 7. Given a term u, we call $st(u)$ the set of its subterms. A recipe $\mathsf{w} \in dom(\phi)$ is said to be *guarded* if the head symbol of $\mathsf{w}\phi$ is an encryption symbol. If w and w' are guarded, then the test $\mathsf{w} = \mathsf{w}'$ is said to be guarded too.

Definition 7 (tests)**.** *Given a protocol P in $\mathcal{C}_{\mathsf{pp}}$, a trace (tr, ϕ) of P and a test $\mathsf{w} = \mathsf{w}'$ where $\mathsf{w}, \mathsf{w}' \in dom(\phi)$, two cases can occur:*

- *$\mathsf{w} = \mathsf{w}'$ is* disjoint*, i.e. $\mathsf{seq}_{\mathsf{tr},\phi}(\mathsf{w})$ and $\mathsf{seq}_{\mathsf{tr},\phi}(\mathsf{w}')$ share no common prefix.*
- *$\mathsf{w} = \mathsf{w}'$ is* forked*, i.e. $\mathsf{seq}_{\mathsf{tr},\phi}(\mathsf{w})$ and $\mathsf{seq}_{\mathsf{tr},\phi}(\mathsf{w}')$ share a common prefix.*

Definition 8 (pulled-up test)**.** *Given a protocol P, a trace (tr, ϕ) of P, a guarded forked test $\mathsf{w} = \mathsf{w}'$ where $\mathsf{w}, \mathsf{w}' \in dom(\phi)$ with maximal common prefix $u = (i_0, \mathsf{w}_{j_0}) \dots (i_p, \mathsf{w}_{j_p})$ is said to be* pulled-up *in (tr, ϕ) if $\mathsf{reset}_{\mathsf{tr},\phi}(\mathsf{w}) = j_p$ where $\mathsf{reset}_{\mathsf{tr},\phi}$ is defined as follows:*

$$\text{if } \mathsf{w}\phi = \mathsf{senc}(u, k, n) \text{ then } \mathsf{reset}_{\mathsf{tr},\phi}(\mathsf{w}) = \min_{1 \le i \le |\phi|}\{i \mid \mathsf{w}\phi \in st(\mathsf{w}_i\phi)\}.$$

Proof. This definition is indeed consistent: by definition of a guarded test, $\mathsf{w}\phi$ is an encryption with some random seed n. Because nonces are uniquely generated, n has to appear in any term $\mathsf{w}_k\phi$ of $\mathsf{seq}_{\mathsf{tr},\phi}(\mathsf{w})$ occuring after $\mathsf{reset}_{\mathsf{tr},\phi}(\mathsf{w})$. Thus $\mathsf{w}\phi$ is a subterm of any such $\mathsf{w}_k\phi$, and in particular of $\mathsf{w}_{\mathsf{reset}_{\mathsf{tr},\phi}(\mathsf{w})}\phi$. □

Moreover, as protocols in $\mathcal{C}_{\mathsf{pp}}$ are deterministic, given a sequence of labels tr and a protocol P, there exists at most one frame ϕ such that $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$.

Lemma 8. *Let P and Q be two procotols in $\mathcal{C}_{\mathsf{pp}}$, if $P \approx_{\mathsf{fwd}} Q$ then for every trace (tr, ϕ) of P and every $\mathsf{w}, \mathsf{w}' \in dom(\phi)$, if $\mathsf{w}\phi = \mathsf{w}'\phi = \mathsf{c}$ for some constant c then $\mathsf{w}\psi = \mathsf{w}'\psi = \mathsf{c}'$ where ψ is the frame such that $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$ and c' is a constant.*

Proof. The only non-trivial point to prove is that if $\mathsf{w}\phi = \mathsf{c}$, then $\mathsf{w}\psi$ has to be a constant. Because protocols in P allow replication for every branch, consider the trace obtained by "replaying" tr in P and Q, *i.e.* given (tr, ϕ), build $(\mathsf{tr}', \phi') \in \mathsf{trace}(P)$ such that every couple input/output in tr appears twice in tr' with the same channel name. If tr is as follows:

$$\mathsf{tr} = \mathsf{in}(c_{i_1}, \mathsf{start}).\mathsf{out}(c_{i_1}, \mathsf{w}_1) \dots \mathsf{in}(c_{i_l}, \mathsf{w}_l).\mathsf{out}(c_{i_l}, \mathsf{w}_m)$$

then tr' is defined as:

$$\begin{cases} \mathsf{tr}' = \mathsf{tr}.\bar{\mathsf{tr}} \\ \bar{\mathsf{tr}} = \mathsf{in}(c_{i_1}, \mathsf{start}).\mathsf{out}(c_{i_1}, \mathsf{w}_{|\phi|+1}) \dots \mathsf{in}(c_{i_l}, \mathsf{w}_{|\phi|+l}).\mathsf{out}(c_{i_l}, \mathsf{w}_{|\phi|+m}) \end{cases}$$

where every occurrence of start in tr is kept in $\bar{\mathsf{tr}}$ but occurrences of w_k are replaced by $\mathsf{w}_{|\phi|+k}$, $|\phi|$ being the cardinal of $dom(\phi)$; and $\mathsf{tr}.\bar{\mathsf{tr}}$ denotes the concatenation of the two sequences of labels, which is a valid trace. We get symmetrically $(\mathsf{tr}', \psi') \in \mathsf{trace}(Q)$. In particular, there exists $\mathsf{w}_* \in dom(\phi')$ such that $\mathsf{w}\phi' = \mathsf{w}_*\phi' = \mathsf{c}$ and $\mathsf{w} = \mathsf{w}_*$ is disjoint (as the entire trace has been duplicated). As $P \approx_{\mathsf{fwd}} Q$, necessarily $\mathsf{w}\psi' = \mathsf{w}_*\psi'$. Because the test is disjoint, if w were guarded, $\mathsf{w}\psi'$ could not share the same top-level random seed as $\mathsf{w}_*\psi'$. Hence $\mathsf{w}\psi$ is a constant. □

Lemma 9. *Let P and Q be two procotols in $\mathcal{C}_{\mathsf{pp}}$, if $P \approx_{\mathsf{fwd}} Q$ then for every trace $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ and every $\mathsf{w}, \mathsf{w}' \in dom(\phi)$, if $\mathsf{w}\phi = \mathsf{w}'\phi$ and $\mathsf{w} = \mathsf{w}'$ is pulled-up in (tr, ϕ) then $\mathsf{w}\psi = \mathsf{w}'\psi$ and $\mathsf{w} = \mathsf{w}'$ is pulled-up in (tr, ψ) where ψ is the frame such that $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$*

Proof. The only non-trivial point to prove is that if $\mathsf{w}\phi = \mathsf{w}'\phi$ and $\mathsf{w} = \mathsf{w}'$ is pulled-up in (tr, ϕ) then $\mathsf{w} = \mathsf{w}'$ is pulled-up in (tr, ψ). We can still assume that $\mathsf{w} = \mathsf{w}'$ is guarded in ψ (it would otherwise contradict Lemma 8). Suppose it is not the case:

1. if $\mathsf{reset}_{\mathsf{tr},\psi}(\mathsf{w}) > \mathsf{reset}_{\mathsf{tr},\phi}(\mathsf{w})$: then $\mathsf{w}\psi \neq \mathsf{w}'\psi$. Indeed, as $\mathsf{reset}_{\mathsf{tr},\psi}(\mathsf{w})$ marks the first index in the frame where the top-level random seed of $\mathsf{w}\psi$ is introduced, having an equality would imply the common prefix of $\mathsf{seq}_{\mathsf{tr},\psi}(\mathsf{w})$ and $\mathsf{seq}_{\mathsf{tr},\psi}(\mathsf{w}')$ to include $\mathsf{w}_{\mathsf{reset}_{\mathsf{tr},\psi}(\mathsf{w})}$ as nonces are uniquely generated, but this prefix is the same as the common prefix of $\mathsf{seq}_{\mathsf{tr},\phi}(\mathsf{w})$ and $\mathsf{seq}_{\mathsf{tr},\phi}(\mathsf{w}')$. Because the test is pulled-up in (tr, ϕ), this prefix ends exactly with $(c_i, \mathsf{w}_{\mathsf{reset}_{\mathsf{tr},\phi}(\mathsf{w})})$ for some channel c_i. Hence $\mathsf{reset}_{\mathsf{tr},\phi}(\mathsf{w}) > \mathsf{reset}_{\mathsf{tr},\psi}(\mathsf{w})$, contradicting the hypothesis.

2. if $\mathsf{reset}_{\mathsf{tr},\psi}(\mathsf{w}) < \mathsf{reset}_{\mathsf{tr},\phi}(\mathsf{w})$: we will show that $P \not\approx_{\mathsf{fwd}} Q$ as there exists a trace $(\mathsf{tr}_*, \psi_*) \in \mathsf{trace}(Q)$, $\mathsf{w}_*, \mathsf{w}'_* \in dom(\psi_*)$ such that $\mathsf{w}_*\psi_* = \mathsf{w}'_*\psi_*$ but $\mathsf{w}_*\phi_* \neq \mathsf{w}'_*\phi_*$, where ϕ_* is defined as expected from our semantics. If such a frame does not exist, it trivially contradicts our hyptothesis. Let s be the maximal common prefix of $\mathsf{seq}_{\mathsf{tr},\psi}(\mathsf{w})$ and $\mathsf{seq}_{\mathsf{tr},\psi}(\mathsf{w}')$, $p = \mathsf{reset}_{\mathsf{tr},\psi}(\mathsf{w})$. Necessarily there exists a channel name c_{i_p} such that $(c_{i_p}, \mathsf{w}_p) \in s$: indeed, as $\mathsf{w} = \mathsf{w}'$ is guarded and true in ψ, their random seeds have to be identical and thus introduced at the same point in the trace. Consequently, there exist three sequences s_1, s_2, s_3 such that $s = s_1.(c_{i_p}, \mathsf{w}_p).s_2$ and $\mathsf{seq}_{\mathsf{tr},\psi}(\mathsf{w}) = s.s_3$. Similarly, there exists s'_3 such that $\mathsf{seq}_{\mathsf{tr},\psi}(\mathsf{w}') = s.s'_3$. From these sequences we can define (tr_*, ψ_*) "corresponding" to $\pi_1(s_1.(c_{i_p}, \mathsf{w}_p).(s_2.s_3))$ where π_1 denotes the generalized projection on the first component; and the according recipes $\mathsf{w}_*, \mathsf{w}'_*$ such that $\mathsf{seq}_{\mathsf{tr}_*,\psi_*}(\mathsf{w}_*) = s_1.(c_{i_p}, \mathsf{w}_p).s_2.s_3$ for some $\mathsf{w}_p \in dom(\psi_*)$ (up to a reindexing of the indices of the variables w_k in s_1,s_2 and s_3) and $\mathsf{seq}_{\mathsf{tr}_*,\psi_*}(\mathsf{w}'_*) = s_1.(c_{i_p}, \mathsf{w}_p).s_2.s'_3 =$ (up to another reindexing of the indices of the variables w_k in s_1,s_2 and s'_3). More precisely, if $\mathsf{seq}_{\mathsf{tr},\psi}(\mathsf{w}') = s_1.(c_{i_p}, \mathsf{w}_p).(c_{i_1}, \mathsf{w}_{j_1}).(c_{i_2}, \mathsf{w}_{j_2}) \dots (c_{i_l}, \mathsf{w}_{j_l})$:

$$\begin{cases} \bar{\mathsf{tr}} & = \mathsf{in}(c_{i_1}, \mathsf{w}_p).\mathsf{out}(c_{i_1}, \mathsf{w}_{|\phi|+1}).\mathsf{in}(c_{i_2}, \mathsf{w}_{|\phi|+1}).\mathsf{out}(c_{i_2}, \mathsf{w}_{|\phi|+2}) \\ & \qquad \dots \mathsf{in}(c_{i_l}, \mathsf{w}_{|\phi|+l-1}).\mathsf{out}(c_{i_l}, \mathsf{w}_{|\phi|+l}) \\ \mathsf{tr}_* & = \mathsf{tr}.\bar{\mathsf{tr}} \end{cases}$$

and ψ_* defined as expected from our semantics. Let $\mathsf{w}_* = \mathsf{w}$ and $\mathsf{w}'_* = \mathsf{w}_{|\phi|+l}$. We can now show that:

(a) $\mathsf{w}_* = \mathsf{w}'_*$ *is pulled-up in* (tr_*, ψ_*): $s_1.(i_p, \mathsf{w}_p)$ is the longest common prefix of $\mathsf{seq}_{\mathsf{tr}_*,\psi_*}(\mathsf{w}_*)$ and $\mathsf{seq}_{\mathsf{tr}_*,\psi_*}(\mathsf{w}'_*)$ by definition of tr_*.

(b) $\mathsf{w}_*\psi_* = \mathsf{w}'_*\psi_*$: by definition of tr_*, $\mathsf{w}'_*\psi_*$ and $\mathsf{w}'\psi$ are already equal up to a renaming of random seeds, as the left components of $\mathsf{seq}_{\mathsf{tr},\psi}(\mathsf{w}')$ and $\mathsf{seq}_{\mathsf{tr}_*,\psi_*}(\mathsf{w}'_*)$ match. As $\mathsf{w}_*\psi_* = \mathsf{w}\psi = \mathsf{w}'\psi$, $\mathsf{w}_*\psi_*$ and $\mathsf{w}'_*\psi_*$ are equal up to a renaming of their random seeds. According to Definition 8, $\mathsf{w}_*\psi_*$ and $\mathsf{w}'_*\psi_*$ are both subterms of $\mathsf{w}_p\psi_*$, hence $\mathsf{w}_*\psi_* = \mathsf{w}'_*\psi_*$.

Finally, as $P \approx_{\mathsf{fwd}} Q$, there exists ϕ_* such that $(\mathsf{tr}_*, \phi_*) \in \mathsf{trace}(P)$. But now $\mathsf{w}_* = \mathsf{w}'_*$ is true and pulled-up in (tr_*, ψ_*) and $\mathsf{reset}_{\mathsf{tr}_*,\psi_*}(\mathsf{w}_*) < \mathsf{reset}_{\mathsf{tr}_*,\phi_*}(\mathsf{w}_*)$ since $\mathsf{reset}_{\mathsf{tr},\psi}(\mathsf{w}) < \mathsf{reset}_{\mathsf{tr},\phi}(\mathsf{w})$. As shown in the previous bullet, it implies that $\mathsf{w}_*\phi_* \neq \mathsf{w}'_*\phi_*$, contradicting the equivalence of P and Q. □

Lemma 10 (characterization of trace equivalence)**.** *Let P and Q be two protocols in $\mathcal{C}_{\mathsf{pp}}$, then $P \approx_{\mathsf{fwd}} Q$ if, and only if, all of the following conditions are true:*

- *for all $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$, there exists a frame ψ such that $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$ and for every $\mathsf{w}, \mathsf{w}' \in dom(\phi)$ and for every constant $\mathsf{c} \in \Sigma_0 \cup \{\mathsf{start}\}$, $\mathsf{w}\phi = \mathsf{w}'\phi = \mathsf{c}$ if, and only if, there exists a constant $\mathsf{c}' \in \Sigma_0 \cup \{\mathsf{start}\}$ such that $\mathsf{w}\psi = \mathsf{w}'\psi = \mathsf{c}'$.* ($\textsc{Const}_P$)
- *and conversely.* ($\textsc{Const}_Q$)
- *for all $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$, there exists a frame ψ such that $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$ and every test that is true and pulled-up in (tr, ϕ) is true and pulled-up in (tr, ψ).* ($\textsc{Fork}_P$)
- *and conversely.* ($\textsc{Fork}_Q$)

Proof. The direct implication is an application of Lemmas 8 and 9.

Let us focus on the converse. Suppose $P \not\approx_{\mathsf{fwd}} Q$. There exists for instance a trace (tr, ϕ) of P such that either there exists no frame ψ such that $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$, in which case conditions $\textsc{Const}_P$ and $\textsc{Fork}_P$ fail, or ψ is indeed defined and there exists a test $\mathsf{w} = \mathsf{w}'$ which holds true in ϕ or ψ but fails in the other. Let us assume that $\mathsf{w}\phi = \mathsf{w}'\phi$ but $\mathsf{w}\psi \neq \mathsf{w}'\psi$.

If $\mathsf{w}\phi = \mathsf{w}'\phi = \mathsf{c}$ for some constant c, then condition $\textsc{Const}_P$ is false.

Else, if the test is guarded, from tr and $\mathsf{w} = \mathsf{w}'$ we will build a new trace (tr_*, ϕ_*) and test $\mathsf{w}_* = \mathsf{w}'_*$ which will be true and pulled-up in (tr_*, ϕ_*).

Let s be the maximal common prefix of $\mathsf{seq}_{\mathsf{tr},\psi}(\mathsf{w})$ and $\mathsf{seq}_{\mathsf{tr},\psi}(\mathsf{w}')$, $p = \mathsf{reset}_{\mathsf{tr},\psi}(\mathsf{w})$. Necessarily there exists a channel name c_{i_p} such that $(c_{i_p}, \mathsf{w}_p) \in s$: indeed, as $\mathsf{w} = \mathsf{w}'$ is guarded and true in ψ, their random seeds have to be identical and thus introduced at the same point in the trace. Consequently, there exist three sequences s_1, s_2, s_3 such that $s = s_1.(c_{i_p}, \mathsf{w}_p).s_2$ and $\mathsf{seq}_{\mathsf{tr},\psi}(\mathsf{w}) = s.s_3$. Similarly, there exists s'_3 such that $\mathsf{seq}_{\mathsf{tr},\psi}(\mathsf{w}') = s.s'_3$. From these sequences we can define (tr_*, ψ_*) "corresponding" to $\pi_1(s_1.(c_{i_p}, \mathsf{w}_p).(s_2.s_3))$ where π_1 denotes the generalized projection on the first component; and the according recipes $\mathsf{w}_*, \mathsf{w}'_*$ such that $\mathsf{seq}_{\mathsf{tr}_*,\psi_*}(\mathsf{w}_*) = s_1.(c_{i_p}, \mathsf{w}_p).s_2.s_3$ for some $\mathsf{w}_p \in dom(\psi_*)$ (up to a reindexing of the indices of the variables w_k in s_1,s_2 and s_3) and $\mathsf{seq}_{\mathsf{tr}_*,\psi_*}(\mathsf{w}'_*) = s_1.(c_{i_p}, \mathsf{w}_p).s_2.s'_3 =$ (up to another reindexing of the

indices of the variables w_k in s_1,s_2 and s_3'). More precisely, if $\mathsf{seq}_{\mathsf{tr},\psi}(\mathsf{w}') = s_1.(c_{i_p},\mathsf{w}_p).(c_{i_1},\mathsf{w}_{j_1}).(c_{i_2},\mathsf{w}_{j_2})\dots(c_{i_l},\mathsf{w}_{j_l})$:

$$\begin{cases} \bar{\mathsf{tr}} & = \mathsf{in}(c_{i_1},\mathsf{w}_p).\mathsf{out}(c_{i_1},\mathsf{w}_{|\phi|+1}).\mathsf{in}(c_{i_2},\mathsf{w}_{|\phi|+1}).\mathsf{out}(c_{i_2},\mathsf{w}_{|\phi|+2}) \\ & \qquad\qquad \dots \mathsf{in}(c_{i_l},\mathsf{w}_{|\phi|+l-1}).\mathsf{out}(c_{i_l},\mathsf{w}_{|\phi|+l}) \\ \mathsf{tr}_* & = \mathsf{tr}.\bar{\mathsf{tr}} \end{cases}$$

and ψ_* defined as expected from our semantics. If $\mathsf{w} = \mathsf{w}_{j_k}$ for some index j_k in ψ, $\mathsf{w}_* = \mathsf{w}$ in ϕ_*, and symmetrically for $\mathsf{w}'_* = \mathsf{w}_{|\phi|+l}$. In particular, we have that $\mathsf{w}\phi = \mathsf{w}_*\phi_*$. Now, either there exists no frame ψ_* such that $(\mathsf{tr}_*,\psi_*) \in \mathsf{trace}(Q)$, in which case condition $\textsc{Fork}_P$ fails obviously, or such a frame exists. In this case, we need then to show that:

1. $\mathsf{w}_* = \mathsf{w}'_*$ *is pulled-up in* (tr_*,ϕ_*): $s_1.(c_{i_p},\mathsf{w}_p)$ is the maximal common prefix of $\mathsf{seq}_{\mathsf{tr}_*,\phi_*}(\mathsf{w}_*)$ and $\mathsf{seq}_{\mathsf{tr}_*,\phi_*}(\mathsf{w}'_*)$ by definition of tr_*.

2. $\mathsf{w}_*\phi_* = \mathsf{w}'_*\phi_*$: by definition of tr_*, $\mathsf{w}'_*\phi_*$ and $\mathsf{w}'\phi$ are already equal up to a renaming of random seeds, as the left components of $\mathsf{seq}_{\mathsf{tr},\phi}(\mathsf{w}')$ and $\mathsf{seq}_{\mathsf{tr}_*,\phi_*}(\mathsf{w}'_*)$ match. As $\mathsf{w}_*\phi_* = \mathsf{w}\phi = \mathsf{w}'\phi$, $\mathsf{w}_*\phi_*$ and $\mathsf{w}'_*\phi_*$ are equal up to a renaming of their random seeds. According to Definition 8, $\mathsf{w}_*\phi_*$ and $\mathsf{w}'_*\phi_*$ are both subterms of $\mathsf{w}_p\phi_*$, hence $\mathsf{w}_*\phi_* = \mathsf{w}'_*\phi_*$.

3. $\mathsf{w}_*\psi_* \neq \mathsf{w}'_*\psi_*$: similarly, we already know that $\mathsf{w}\psi = \mathsf{w}_*\psi_*$. Suppose *ad absurdum* that $\mathsf{w}_*\psi_* = \mathsf{w}'_*\psi_*$. Because the left components of $\mathsf{seq}_{\mathsf{tr},\psi}(\mathsf{w}')$ and $\mathsf{seq}_{\mathsf{tr}_*,\psi_*}(\mathsf{w}'_*)$ match, $\mathsf{w}'\psi$ and $\mathsf{w}'_*\psi_*$ are either constant and equal or cyphertexts and equal up to a renaming of their random seeds. In the first case, it is enough to conclude that $\mathsf{w}\psi = \mathsf{w}'\psi$, which is absurd. In the second case, $\mathsf{w}_*\psi_*$ and $\mathsf{w}'_*\psi_*$ being randomized, must have equal top-level random seeds, implying that this nonce was introduced before (c_{i_p},w_p) in the common prefix of their respective sequences. As the said prefix is also common, up to a reindexing of the w_k, to w and w', $\mathsf{w}\psi$ and $\mathsf{w}'\psi$ share the same top-level random seed and are thus equal, contradicting our hypothesis. Threfore: $\mathsf{w}_*\psi_* \neq \mathsf{w}'_*\psi_*$.

Hence $\textsc{Fork}_P$ is false. Finally if $\mathsf{w}\psi = \mathsf{w}'\psi$ but $\mathsf{w}\phi \neq \mathsf{w}'\phi$, conditions $\textsc{Fork}_Q$ will similarly fail. □

From this point on, we focus on building two classes automata, $\mathcal{A}_{\textsc{Const}}$ and $\mathcal{A}_{\textsc{Fork}}$, whose language equivalence is itself equivalent to the properties introduced in Lemma 10. In order to use terms in the stack of an automaton, we introduce the following representation: given a term u we define inductively $\bar{u}$ in the following way:

$$\begin{cases} \bar{u} = \bar{v}.\mathsf{k} & \text{if } u = \mathsf{f}(v,\mathsf{k},n) \text{ and } \mathsf{f} \in \{\mathsf{senc},\mathsf{aenc},\mathsf{sign}\} \\ \bar{\mathsf{c}} = \mathsf{c} & \text{for any constant } c \\ \bar{x} = \epsilon & \text{for any variable } x \end{cases}$$

where ϵ denotes the empty word.

Transitions are often written in a clearer way in the following fashion: a transition from q to q' reading a, popping $\bar{u}$ from the stack and pushing $\bar{v}$ will be denoted by $q \xrightarrow{a;\bar{u}/\bar{v}} q'$.

Lemma 11. *Let P and Q be two protocols in $\mathcal{C}_{\mathsf{pp}}$, there exist four real-time GPDA $\mathcal{A}_c^R$ for R in $\{P,Q\}$ and c in $\{\textsc{Const},\textsc{Fork}\}$ such that:*

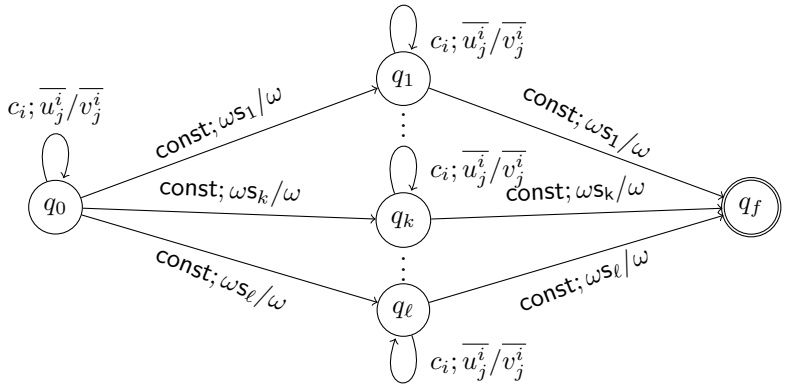


Figure 1: Automaton $\mathcal{A}^P_{\text{Const}}$

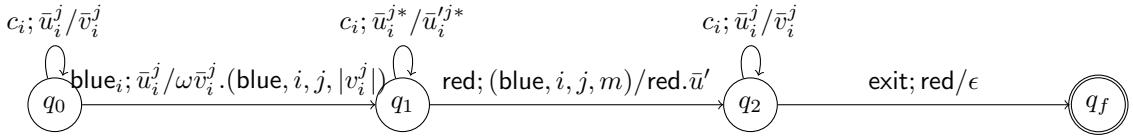


Figure 2: Automaton $\mathcal{A}^P_{\text{Fork}}$

- *P and Q satisfy conditions* Const_P *and* Const_Q *iff* $\mathcal{L}(\mathcal{A}^P_{\text{Const}}) = \mathcal{L}(\mathcal{A}^Q_{\text{Const}})$
- *P and Q satisfy conditions* Fork_P *and* Fork_Q *iff* $\mathcal{L}(\mathcal{A}^P_{\text{Fork}}) = \mathcal{L}(\mathcal{A}^Q_{\text{Fork}})$.

Proof. In the following, we denote by Σ_0^P (resp. Σ_0^Q) the finite set of constants of $\Sigma_0 \cup \{\mathsf{start}\}$ that actually occur in P (resp. Q). First, let us define $\mathcal{A}^P_{\text{Const}}$, schematized in Figure 1:

$$\mathcal{A}^P_{\text{Const}} = (\{q_0, q_f\} \cup \{q_\mathsf{s} \text{ for every constant } \mathsf{s}\}, \Pi, \Gamma, q_0, \omega, \{q_f\}, \delta)$$

where

- $\Gamma = \Sigma_0^P$,
- $\Pi = \{c_1, \ldots, c_n\} \cup \{\mathsf{const}\}$
- δ is defined as follows:
  - **From q_0 to q_0:** for every $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, p_i\}$, there exists a transition $(c_i; \bar{u}_i^j/\bar{v}_i^j)$ if v_i^j is *not* ground, or a transition $(c_i; \bar{u}_i^j/\omega\bar{v}_i^j)$ if v_i^j is ground.
  - **From q_0 to q_s for every constant s:** there exists a transition $(\mathsf{const}; \omega\mathsf{s}/\omega)$.
  - **From q_s to q_s for every constant s:** for every $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, p_i\}$, there exists a transition $(c_i; \bar{u}_i^j/\bar{v}_i^j)$ if v_i^j is *not* ground, or a transition $(c_i; \bar{u}_i^j/\omega\bar{v}_i^j)$ if v_i^j is ground.

– **From q_{s} to q_f for every constant s:** there exists a transition $(\mathsf{const}; \omega\mathsf{s}/\omega)$.

Moreover we need to add in states q_0 and q_{s} a transition $(c_{\mathsf{start}}, \omega/\omega.\mathsf{start})$ to put the constant start on the stack at first. The loops in q_0 and q_{s} are meant to represent the construction of a term through a regular execution of the protocol in a trace, unstacking as terms are filtered and stacking when outputting. The transitions labeled $(\mathsf{const}; \omega\mathsf{s}/\omega)$ are activated when a constant is reached and enable the attacker to "memorize" which constant she reached, and to reach it again in order to exhibit an equality between two terms of a frame.

$\mathcal{A}^P_{\text{CONST}}$ has a number of states polynomial in the number of constants in P, and a number of transitions similar to the number of branches times the maximal length of any term occuring in P. Thus, $\mathcal{A}^P_{\text{CONST}}$ is of size polynomial with respect to the size of P.

We need to prove that P and Q satisfy conditions CONST_P and CONST_Q if, and only if, $\mathcal{L}(\mathcal{A}^P_{\text{CONST}}) = \mathcal{L}(\mathcal{A}^Q_{\text{CONST}})$.

1. If $\mathcal{L}(\mathcal{A}^P_{\text{CONST}}) \neq \mathcal{L}(\mathcal{A}^Q_{\text{CONST}})$: suppose there exists a word $u \in \mathcal{L}(\mathcal{A}^P_{\text{CONST}}) \setminus \mathcal{L}(\mathcal{A}^Q_{\text{CONST}})$. Let us build a trace $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ "corresponding" to u: let k^1_c and k^2_c be the positions in u of the two occurences of const and u_0 be u stripped from these const. If $u_0 = c_{i_1}.c_{i_2} \dots c_{i_l}$, let us define tr as follows:

$$\mathsf{tr} = \quad \mathsf{in}(c_{i_1}, \mathsf{start}).\mathsf{out}(c_{i_1}, \mathsf{w}_1) \dots \mathsf{in}(c_{i_l}, \mathsf{w}_{l-1}).\mathsf{out}(c_{i_l}, \mathsf{w}_l)$$

and ϕ is defined uniquely as expected from our semantics, as P is deterministic. (tr, ϕ) is indeed a valid trace of P as the transition function δ fully captures input filtering and output of terms for protocols in $\mathcal{C}_{\mathsf{pp}}$. We can now define $\mathsf{w} = \mathsf{w}_{k^1_c - 1}$ and $\mathsf{w}' = \mathsf{w}_{k^2_c}$. Because the transitions from q_0 to q_{c} and then from q_{c} to q_f for some constant c were possible, we get that $\mathsf{w}\phi = \mathsf{w}'\phi = \mathsf{c}$. As $u \notin \mathcal{L}(\mathcal{A}^Q_{\text{CONST}})$, let $q \xrightarrow{a;\alpha/\beta} q'$ be the first failing transition in the run of u in $\mathcal{A}^Q_{\text{CONST}}$: if $q = q'$, then $a = c_i$ for some $i \in \{1, \dots, n\}$, meaning that there exists no frame ψ such that $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$: CONST_Q fails. On the other hand, if $q = q_0$, $q' = q_{\mathsf{c}}$ for some constant c, then $\mathsf{w}\psi \neq \mathsf{c}$ for any constant c: $\mathsf{w}\psi$ is thus a ciphertext, contradicting CONST_Q. Last, if $q = q_{\mathsf{c}}$ and $q' = q_f$ for some constant c, then $\mathsf{w}\psi = \mathsf{c}$ but $\mathsf{w}\psi \neq \mathsf{c}$, making CONST_Q fail once again. Hence P ans Q do not satisfy CONST_Q. Symmetrically, if $u \in \mathcal{L}(\mathcal{A}^Q_{\text{CONST}}) \setminus \mathcal{L}(\mathcal{A}^P_{\text{CONST}})$, CONST_P is false.

2. If P and Q do not satisfy CONST_P nor CONST_Q: suppose CONST_Q fails, *i.e.* there exists a trace (tr, ϕ) of P, $\mathsf{w}, \mathsf{w}' \in dom(\phi)$ and a constant c such that $\mathsf{w}\phi = \mathsf{w}'\phi = \mathsf{c}$ but:

   - either there exists no frame ψ of Q such that $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$
   - or $\mathsf{w}\psi$ is not a constant
   - or $\mathsf{w}\psi$ is a constant but $\mathsf{w}\psi \neq \mathsf{w}'\psi$

   From tr we build a word $u \in \mathcal{L}(\mathcal{A}^P_{\text{CONST}})$: $u = u_1.\mathsf{const}.u_2.\mathsf{const}$ with

$$u_1 = \pi_1(\mathsf{seq}_{\mathsf{tr},\phi}(\mathsf{w}))$$
$$u_2 = \pi_1(\mathsf{seq}_{\mathsf{tr},\phi}(\mathsf{w}'))$$

where $\pi_1(x)$ denotes the generalised projection on the first component of the sequence x. As the transition function δ fully captures input filtering and output of terms for protocols in $\mathcal{C}_{\mathsf{pp}}$, we get that upon reading the first const, $\mathcal{A}^P_{\textsc{Const}}$ is in q_0, the transition $q_0 \xrightarrow{\mathsf{const};\omega\mathsf{c}/\omega} q_{\mathsf{c}}$ is indeed possible as $\mathsf{w}\phi = \mathsf{c}$; and similarly upon reading the second const, $\mathcal{A}^P_{\textsc{Const}}$ is in q_{c}, the transition $q_{\mathsf{c}} \xrightarrow{\mathsf{const};\omega\mathsf{c}/\omega} q_f$ is indeed possible as $\mathsf{w}'\phi = \mathsf{c}$, hence $u \in \mathcal{L}(\mathcal{A}^P_{\textsc{Const}})$. What remains to show is that $u \notin \mathcal{L}(\mathcal{A}^Q_{\textsc{Const}})$. Three cases occur, corresponding to the previous highlighted situations:

- if (tr, ψ) can never be a valid trace of Q, a letter $c_i \in \{1, \ldots, n\}$ will not be read in state q_0 or q_{c} as the stack does not contain the adequate symbols to enable the transition, reflecting the failed filtering on the input on this channel
- if $\mathsf{w}\psi$ is not a constant : no transition $q_0 \xrightarrow{\mathsf{const};\omega\mathsf{c}/\omega} q_{\mathsf{c}}$ will be possible after u_1
- if $\mathsf{w}\psi$ is a constant c but $\mathsf{w}\psi \neq \mathsf{w}'\psi$: the transition $q_0 \xrightarrow{\mathsf{const};\omega\mathsf{c}/\omega} q_c$ will not be possible after u_2.

Hence u cannot belong to $\mathcal{L}(\mathcal{A}^Q_{\mathsf{const}})$.

Therefore P and Q satisfy conditions $\textsc{Const}_P$ and $\textsc{Const}_Q$ iff $\mathcal{L}(\mathcal{A}^P_{\textsc{Const}}) = \mathcal{L}(\mathcal{A}^Q_{\textsc{Const}})$.

Finally, let us define $\mathcal{A}^P_{\textsc{Fork}}$, schematized in Figure 2:

$$\mathcal{A}^P_{\textsc{Fork}} = (\{q_0, q_1, q_2, q_f\}, \Pi, \Gamma, q_0, \omega, \{q_f\}, \delta)$$

where

- $\Gamma = \Sigma_0^P \cup \{\mathsf{red}\} \cup (\mathsf{blue} \times \{1, \ldots, n\} \times \{1, \ldots, \max_i(p_i)\} \times \{1, \ldots, \max_{i,j}|v_i^j|\})$
- $\Pi = \{c_1, \ldots, c_n\} \cup \{\mathsf{blue}_i\}_{1 \leq i \leq n} \cup \{\mathsf{red}, \mathsf{exit}\}$
- δ is defined as follows:
  - **From q_0 to q_0:** for every $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, p_i\}$, there exists a transition $(c_i; \bar{u}_i^j/\bar{v}_i^j)$ if v_i^j is *not* ground, or a transition $(c_i; \bar{u}_i^j/\omega.\bar{v}_i^j)$ if v_i^j is ground.
  - **From q_0 to q_1:** for every $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, p_i\}$, there exists a transition $(c_i; \bar{u}_i^j/\omega.\bar{v}_i^j.(\mathsf{blue}, i, j, |v_i^j|))$.
  - **From q_1 to q_1:** for every $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, p_i\}$, for every $m \in \{1, \ldots, \max_{i,j}|v_i^j|\}$ and for every suffix $\bar{v}$ of $\bar{u}_i^j$ of length $k < m$ such that $\bar{u}_i^j = \bar{w}.\bar{v}$ there exists a transition $(c_i; \bar{w}.(\mathsf{blue}, i, j, m).\bar{v}/(\mathsf{blue}, i, j, m-k)\bar{v}_i^j)$, in addition to the same transitions as from q_0 to q_0.
  - **From q_1 to q_2:** for every $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, p_i\}$, for every $m \in \{1, \ldots, |v_i^j|\}$, there exists a transition $(\mathsf{red}; (\mathsf{blue}, i, j, m)/\mathsf{red}.u')$ where $\bar{u}'$ is the suffix of length $|v_i^j| - m$ of $\bar{v}_i^j$.

- **From q_2 to q_2:** for every $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, p_i\}$, there exists a transition $(c_i; \bar{u}_i^j / \bar{v}_i^j)$ if v_i^j is *not* ground, or a transition $(c_i; \bar{u}_i^j / \omega.\bar{v}_i^j)$ if v_i^j is ground.
- **From q_2 to q_f:** there exists a transition $(\mathsf{exit}; \mathsf{red}/\epsilon)$.

As in the previous automata, we need to add in states q_0 a transition $(c_{\mathsf{start}}, \omega/\omega\mathsf{start})$ to put the constant start on the stack at first. The loop in q_0 represents the regular execution of the protocol by the attacker: through unstacking and stacking, she builds a term on the stack along a particular trace. The transition $(\mathsf{blue}_i; z/z')$ enable her to mark a fork when building a test in her frame with a particular stack symbol (tile) blue, enriched with information about how and where it was put on the stack. This fork corresponds to the $\mathsf{reset}_{\mathsf{tr},\phi}$ operator defined sooner. By looping in q_1, the attacker can continue building the first term of an equality, following the usual execution of the protocol, if it were for the presence of the new tile blue which can only go down on the stack for at most $|u_i^j|$ tiles. If this tile appears on top of the stack, meaning the attacker builds a pulled-up test, she can try to go back to her mark to build the second member of the test. If this second term manages to end up exactly as the previous one, an equality is reached and the word is recognized by the automata, witnessing the said equality.

Note that $\mathcal{A}^P_{\text{Fork}}$ has a fixed number of states, and a polynomial number of transitions : transitions are added for each branch and suffix of any input term. Thus, $\mathcal{A}^P_{\text{Fork}}$ is of size polynomial with respect to the size of P.

What remains now is to prove that P and Q satisfy conditions Fork_P and Fork_Q if, and only if, $\mathcal{L}(\mathcal{A}^P_{\text{Fork}}) = \mathcal{L}(\mathcal{A}^Q_{\text{Fork}})$.

1. If $\mathcal{L}(\mathcal{A}^P_{\text{Fork}}) \neq \mathcal{L}(\mathcal{A}^Q_{\text{Fork}})$: from a word u in $\mathcal{L}(\mathcal{A}^P_{\text{Fork}}) \setminus \mathcal{L}(\mathcal{A}^Q_{\text{Fork}})$ for instance, build the corresponding minimal trace (tr, ϕ) in number of outputs, and a true and pulled-up test $\mathsf{w}_1 = \mathsf{w}_2$. This operation is done in the same fashion as with automaton $\mathcal{A}^P_{\text{Const}}$, by using letters of u as channel names for the couples of input and ouput. The test is indeed pulled-up as a blue tile indicates the first time the top-level random seed of $\mathsf{w}\psi$ appears in the frame. The test is moreover true as activating the last transitions $(\mathsf{exit}; \mathsf{red}/\epsilon)$ requires the stack to be identical to the stack before turning the blue tile red. Several cases for u not belonging to $\mathcal{A}^Q_{\text{Fork}}$ can occur:
   (a) a transition $q_0 \xrightarrow{c_i; z/z'} q_0$ or $q_0 \xrightarrow{\mathsf{blue}_i; z/z'} q_1$ is not possible: there is no frame ψ such that $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$, as these transitions mimick the semantics of the protocol. Thus Fork_Q fails.
   (b) a transition $q_1 \xrightarrow{c_i; \bar{u}_i^{j*} / \bar{u}_i'^{j*}} q_1$ is not possible. Either the previous remark still applies and Q cannot execute a particular input; or a blue tile cannot be unstacked, meaning that the corresponding test will not be pulled-up in (tr, ψ). Indeed it corresponds to the case where $\mathsf{reset}_{\mathsf{tr},\psi}(\mathsf{w}_1) < \mathsf{reset}_{\mathsf{tr},\phi}(\mathsf{w}_1)$. The index in the frame where the blue tile was put, *ie.* the forking point, did not correspond to the first occurence of the top-level random seed of $\mathsf{w}_1\psi$ but to a latter step. $\mathsf{w}_1 = \mathsf{w}_2$ is not pulled-up in (tr, ψ) and Fork_Q is false.
   (c) the blue tile is not at the top of the stack upon becoming red: the corresponding test will not be pulled-up in (tr, ψ) either. It corresponds

to the case where $\mathsf{reset}_{\mathsf{tr},\psi}(\mathsf{w}_1) > \mathsf{reset}_{\mathsf{tr},\phi}(\mathsf{w}_1)$. The forking point did not correspond to the first occurence of the top-level random seed of $\mathsf{w}_1\psi$ but to a previous step. Once again $\mathsf{w}_1 = \mathsf{w}_2$ is not pulled-up in (tr, ψ) and FORK_Q is false.

(d) the red tile is not at the top of the stack upon reading the last letter of the word: the test is pulled-up but false in ψ. The stack at this point, without the red tile, is not identical to the stack before the blue tile turning red, making FORK_Q fail.

Hence FORK_Q fails as soon as $u \notin \mathcal{L}(\mathcal{A}^Q_{\text{FORK}})$. Symmetrically, if $u \in \mathcal{L}(\mathcal{A}^Q_{\text{FORK}}) \setminus \mathcal{L}(\mathcal{A}^P_{\text{FORK}})$, FORK_P is false.

2. If P and Q do not satisfy conditions FORK_P nor FORK_Q. Suppose FORK_Q is false: there exists a trace (tr, ϕ) of P, $\mathsf{w}_1, \mathsf{w}_2 \in dom(\phi)$ such that $\mathsf{w}_1 = \mathsf{w}_2$ is true and pulled-up in (tr, ϕ) but

   - either there exists no frame ψ of Q such that $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$
   - or $\mathsf{w}_1 = \mathsf{w}_2$ is not pulled-up in (tr, ψ)
   - or $\mathsf{w}_1 = \mathsf{w}_2$ is pulled-up in (tr, ψ) but $\mathsf{w}_1\psi \neq \mathsf{w}_2\psi$.

   In the first case, from tr we can build a word $u \in \mathcal{L}(\mathcal{A}^P_{\text{FORK}}) \setminus \mathcal{L}(\mathcal{A}^Q_{\text{FORK}})$: indeed there exists $\mathsf{w} \in dom(\phi)$ such that $\mathsf{in}(c_i, \mathsf{w})$ for some channel c_i is possible in P but not in Q. From $\mathsf{seq}_{\mathsf{tr},\phi}(\mathsf{w})$ build a word $u_1 = u_1'.a$ in the same fashion as with $\mathcal{A}^P_{\text{CONST}}$. Then let $u = u_1'.\mathsf{blue}_a.\mathsf{red}.\mathsf{exit}$: $u \in \mathcal{L}(\mathcal{A}^P_{\text{FORK}}) \setminus \mathcal{L}(\mathcal{A}^Q_{\text{FORK}})$ as $\mathcal{A}^Q_{\text{FORK}}$ cannot evolve from state q_0 as our semantics are consistent with the transitions $(c_i; \bar{u}_i^j/\bar{v}_i^j)$. Let us now consider the two remaining cases:

   (a) $\mathsf{w}_1 = \mathsf{w}_2$ is pulled-up in (tr, ϕ) but not in (tr, ψ): from $\mathsf{seq}_{\mathsf{tr},\phi}(\mathsf{w}_1)$ and $\mathsf{seq}_{\mathsf{tr},\phi}(\mathsf{w}_2)$, as in the previous steps, build a word u which will be in $\mathcal{L}(\mathcal{A}^P_{\text{FORK}})$ in the following way: if $\mathsf{seq}_{\mathsf{tr},\phi}(\mathsf{w}) = s.s_1$ and $\mathsf{seq}_{\mathsf{tr},\phi}(\mathsf{w}') = s.s_1'$ where s is the maximal common prefix of those two sequences, let $u_0.c_m = \pi_1(s)$, $u_1 = \pi_1(s_1)$ and $u_1' = \pi_1(s_1')$. Then $u = u_0.\mathsf{blue}_m.u_1.\mathsf{red}.u_2.\mathsf{exit}$: Then either $\mathsf{reset}_{\mathsf{tr},\phi}(\mathsf{w}_1) < \mathsf{reset}_{\mathsf{tr},\psi}(\mathsf{w}_1)$ in which case the transition $q_1 \xrightarrow{\mathsf{red};(\mathsf{blue},i,j,m)/\mathsf{red}.\bar{u}'} q_2$ is not possible or $\mathsf{reset}_{\mathsf{tr},\phi}(\mathsf{w}_1) > \mathsf{reset}_{\mathsf{tr},\psi}(\mathsf{w}_1)$: a transition $q_1 \xrightarrow{c_i;\bar{u}_i^{j*}/\bar{u}_i'^{j*}} q_1$ or $q_2 \xrightarrow{c_i;\bar{u}_i^j/\bar{v}_i^j} q_2$ will be impossible; the two cases representing whether the top-level random seed in $\mathsf{w}_1\psi$ is introduced (strictly) before the forking occuring when stacking a blue tile.

   (b) $\mathsf{w}_1 = \mathsf{w}_2$ is pulled-up in (tr, ϕ) and in (tr, ψ) but $\mathsf{w}\psi \neq \mathsf{w}'\psi$. Build the word u from the sequences leading to w_1 and w_2 as usual, u will be in $\mathcal{L}(\mathcal{A}^P_{\text{FORK}})$ but not in $\mathcal{L}(\mathcal{A}^Q_{\text{FORK}})$ as the transition $q_2 \xrightarrow{\mathsf{exit};\mathsf{red}/\epsilon} q_f$ is impossible because the tile red will not be at the top of the stack: if it were, the final stack and the one before the transition from q_1 to q_2, minus the blue tile, would have been identical, thus witnessing an equality between $\mathsf{w}_1\psi$ and $\mathsf{w}_2\psi$.

□

We finally get the desired result:

Theorem 3. *Let P and Q in $\mathcal{C}_{\mathsf{pp}}$, we have that: $P \approx_{\mathsf{fwd}} Q \iff \mathcal{L}(\mathcal{A}_P) = \mathcal{L}(\mathcal{A}_Q)$.*

Proof. We define $\mathcal{A}_P$ with an enriched input alphabet such that

$$\mathcal{L}(\mathcal{A}_P) = a.\mathcal{L}(\mathcal{A}^P_{\text{CONST}}) \cup b.\mathcal{L}(\mathcal{A}^P_{\text{FORK}})$$

where a and b are two new symbols. Doing similarly with $\mathcal{A}_Q$, we get using Lemmata 10 and 11 that $P \approx_{\mathsf{fwd}} Q$, if, and only, if $\mathcal{L}(\mathcal{A}_P) = \mathcal{L}(\mathcal{A}_Q)$. □

D From language equivalence to trace equivalence

This section is devoted to the proof of Proposition 2 introduced in Section 5.

Proposition 2. *Let $\mathcal{A}$ and $\mathcal{B}$ be two real-time GPDA: $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B}) \iff P_{\mathcal{A}} \sqsubseteq P_{\mathcal{B}}$.*

Proof. Let $\mathcal{A} = (Q, \Pi, \Gamma, q_0, \omega, Q_f, \delta)$ be real-time GPDA. For the purpose of this encoding, we consider that we have the following constants in Σ_0:

- $\{q \mid q \in Q\}$, i.e. one constant for each state of the automaton;
- $\{\alpha \mid \alpha \in \Gamma\}$, i.e. one constant for each letter in the stack-alphabet;
- an additional constant $\mathsf{k}_{\mathsf{well}}$.

Words in Γ^*, *i.e.* stacks, will be represented through nested symmetric encryption with private keys representing their counterparts in Γ. For the sake of brevity, given a word $u = \alpha_1 \dots \alpha_p$ of Γ^*, we denote by $x.u$:

- either the term $\mathsf{senc}(\dots \mathsf{senc}(x, \alpha_1, y_1) \dots, \alpha_p, y_p)$ where y_1 through y_p are variables used for nonces when $x.u$ is used in as an input pattern;
- or the term $\mathsf{senc}(\dots \mathsf{senc}(x, \alpha_1, n_1) \dots, \alpha_p, n_p)$ where n_1 through n_p are actual names.

For every $a \in \Pi$, $q \in Q$, and $k \in \mathbb{N}$, we define:

$$\begin{cases} U_{q,a} = \{u \in \Gamma^* \mid (q, a, u) \in dom(\delta)\} \\ U^{\mathsf{suf}}_{q,a} = \{u \in \Gamma^* \mid exists v \in U_{q,a}, u \prec v\} \\ \bar{U}_{q,a} = (\omega.U^{\mathsf{suf}}_{q,a} \cup \Gamma.U^{\mathsf{suf}}_{q,a}) \setminus (U_{q,a} \cup U^{\mathsf{suf}}_{q,a}) \end{cases}$$

where $\prec$ denotes the strict suffix relation. Note that because $dom(\delta)$ is finite, as the automaton is finitely described, the sets $U_{q,a}$ and $\bar{U}_{q,a}$ are also finite for any $a \in \Pi$ and $q \in Q$. Moreover, the automaton being deterministic, given $q \in Q$ and $a \in \Pi$, for every word $u \in \omega.\Gamma^*$, either there exists a unique suffix u' of u such that $u \in U_{q,a}$ or there exists a unique suffix u' of u such that $u' \in \bar{U}_{q,a}$, and this disjunction is exclusive.

These sets can be interpreted as follows: $U_{q,a}$ is the set of words on the stack alphabet that can be unstacked upon reading a in state q. $\bar{U}_{q,a}$ can be seen as the set of shortest words which are not suffixes of any word of $U_{q,a}$, and, thus the shortest words to unstack to be sure no transition from q reading a is possible.

Given the RGDPA $\mathcal{A}$, we now define the protocol P in the following way:

$$\begin{array}{rlll} P_{\mathcal{A}} := & ! & \mathsf{in}(c_0, \mathsf{start}).\mathsf{new}\ n.\mathsf{out}(c_0, \mathsf{senc}(\omega, q_0, n)) & (0) \\ | & ! & \mathsf{in}(c_a, \mathsf{senc}(x.u, q, y)).\mathsf{new}\ \tilde{n}.\mathsf{out}(c_a, \mathsf{senc}(x.v, q', n)) & (1_{q,a,u}) \\ | & ! & \mathsf{in}(c_a, \mathsf{senc}(x.u', q, y)).\mathsf{new}\ \tilde{n}.\mathsf{out}(c_a, \mathsf{senc}(x.v, \mathsf{k_{well}}, n)) & (2_{q,a,u'}) \\ | & ! & \mathsf{in}(c_a, \mathsf{senc}(x, \mathsf{k_{well}}, y)).\mathsf{new}\ n.\mathsf{out}(c_a, \mathsf{senc}(x, \mathsf{k_{well}}, n)) & (3_{a,\mathsf{well}}) \\ | & ! & \mathsf{in}(c_f, \mathsf{senc}(x, q_f, y)).\mathsf{out}(c_f, \mathsf{start}) & (4_{q_f}) \end{array}$$

where a quantifies over Π, q over Q, u over the words in $U_{q,a}$, u' over $\bar{U}_{q,a}$ and q_f over Q_f. q' is the first component of $\delta(q,a,u)$, whereas v is its second component. As $\mathcal{A}$ is deterministic, $P_{\mathcal{A}}$ happens to be in $\mathcal{C}_{\mathsf{pp}}$.

We define $P_{\mathcal{B}}$ in a similar fashion with $\mathcal{B}$. We need to prove now that $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ if, and only if, $P_{\mathcal{A}} \sqsubseteq P_{\mathcal{B}}$.

1. If $\mathcal{L}(\mathcal{A}) \not\subseteq \mathcal{L}(\mathcal{B})$, then $P_{\mathcal{A}} \not\sqsubseteq P_{\mathcal{B}}$: there exists a word $w \in \mathcal{L}(\mathcal{A}) \setminus \mathcal{L}(\mathcal{B})$. We will build a trace (tr, ϕ) of $P_{\mathcal{A}}$ such that there exists no equivalent trace (tr, ϕ') of $P_{\mathcal{B}}$. To build (tr, ϕ), we will mimick the behaviour of $\mathcal{A}$ when reading w: the first branches to use is (0), enabling the attacker to activate other branches of the process. As $w \in \mathcal{L}(\mathcal{A})$ and $\mathcal{A}$ is deterministic, there exists a unique sequence of transitions of the form $\delta(q,a,u)$ leading to a accepting state $q_f \in Q_f$. For every such transition the attacker will activate a branch $(1_{q,a,u})$ in $P_{\mathcal{A}}$. If $w = a_1 \dots a_n$, we define (tr, ϕ) as follows:

   $$\begin{array}{ll} \mathsf{tr} = & \mathsf{in}(c_0, \mathsf{start}).\mathsf{out}(c_0, \mathsf{w}_1).\mathsf{in}(c_{a_1}, \mathsf{w}_1).\mathsf{out}(c_{a_1}, \mathsf{w}_2) \\ & \dots \mathsf{in}(c_{a_n}, \mathsf{w}_n).\mathsf{out}(c_{a_n}, \mathsf{w}_{n+1}).\mathsf{in}(c_f, \mathsf{w}_{n+1}).\mathsf{out}(c_f, \mathsf{w}_{n+2}) \end{array}$$

   and ϕ is defined as expected given our semantics. Because of the definition of $U_{q,a}$ for any a and q, the inputs on the channels c_{a_i} are possible, the stack of the automaton upon reading a_i and its current state being faithfully represented by the term $\mathsf{w}_i\phi$. Thus (tr, ϕ) is indeed a trace of $P_{\mathcal{A}}$. When reaching q_f, the attacker can use the branch (4_{q_f}) to output the public constant start. As $w \notin \mathcal{L}(\mathcal{B})$, the corresponding sequence of transitions in $\mathcal{B}$ does not lead to any accepting state:

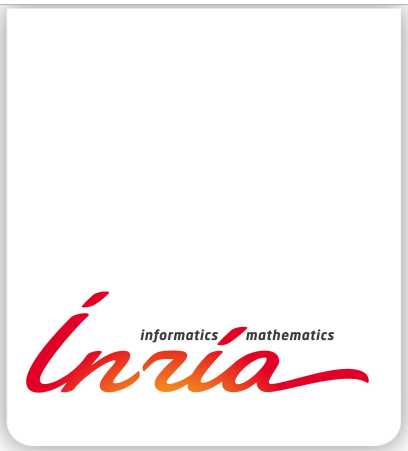
   - either a transition $\delta(q,a,u)$ is not possible, in which case $u \notin U_{q,a}$: then there exists a suffix u' of u such that $u' \in \bar{U}_{q,a}$, enabling a transition $(2_{q,a,u'})$ on channel c_a for the attacker, and every subsequent transition is of the form $(3_{a',\mathsf{well}})$ for some channel $c_{a'}$,
   - or the state reached in $\mathcal{B}$ is not in Q'_f or, as previously described, a transition $(2_{q,a,u'})$ has been taken: the sequence $\mathsf{in}(c_f, \mathsf{w}_{n+1}).\mathsf{out}(c_f, \mathsf{w}_{n+2})$ cannot occur in $P_{\mathcal{B}}$.

   Consequently, there exists no trace $(\mathsf{tr}, \phi') \in \mathsf{trace}(P')$, thus $P_{\mathcal{A}} \not\sqsubseteq P_{\mathcal{B}}$.

2. If $P_{\mathcal{A}} \not\sqsubseteq P_{\mathcal{B}}$, then $\mathcal{L}(\mathcal{A}) \not\subseteq \mathcal{L}(\mathcal{B})$. First note that, for every frame ϕ of $P_{\mathcal{A}}$ or $P_{\mathcal{B}}$, for every $\mathsf{w} \in dom(\phi)$, $\mathsf{w}\phi = \mathsf{start}$ or $\mathsf{w}\phi$ is an encryption with a fresh nonce. In particular, if $\mathsf{w}, \mathsf{w}' \in dom(\phi)$ and $\mathsf{w} \neq \mathsf{w}'$, then $\mathsf{w}\phi = \mathsf{w}'\phi = \mathsf{start}$ or $\mathsf{w}\phi \neq \mathsf{w}'\phi$, meaning that no equality other than one using constant start can appear in ϕ. Now consider the shortest trace (tr, ϕ) of $P_{\mathcal{A}}$, in terms of number of transitions, such that there exists no equivalent frame $(\mathsf{tr}, \psi) \in \mathsf{trace}(P_{\mathcal{B}})$. Because of the definitions of $U_{q,a}$ and $\bar{U}_{q,a}$, for any $a \in \Pi$, a transition of channel c_a is always possible. Two cases can *a priori* occur:

- tr contains no input/output on channel c_f. Then there exists a unique frame ψ such that $(\mathsf{tr}, \psi) \in \mathsf{trace}(P_{\mathcal{B}})$. In such a case, we know that, for every $\mathsf{w} \in dom(\phi)$, $\mathsf{w}\phi \neq \mathsf{start}$. Thus, as mentioned shortly before, no equality whatsoever can hold true in ϕ or ψ. For this reason, ϕ and ψ are statically equivalent, contradicting the non-equivalence of (tr, ϕ) and (tr, ψ).
- or tr contains an input/output on channel c_f. Let $\mathsf{w} \in dom(\phi)$ be the corresponding variable in the frame ϕ. If that output can be matched in $P_{\mathcal{B}}$, then tr is not a minimal witness of non-inclusion as start cannot be used to trigger any other branch. Hence $P_{\mathcal{B}}$ has to be unable to match the input on channel c_f. Let $s = \mathsf{seq}_{\mathsf{tr},\phi}(\mathsf{w})$ and u be the generalised projection of s on its first component. u is then a sequence of channel names. Because tr was the shortest sequence of transitions, $u = c_0.c_{a_1} \dots c_{a_n}.c_f$. Let $v = a_1 \dots a_n$: v is a word of Π^*, and, in particular,
  - $v \in \mathcal{L}(\mathcal{A})$: because transitions $(1_{q,a,u})$ in $P_{\mathcal{A}}$ faithfully represent transitions $\delta(q, a, u)$ in $\mathcal{A}$ and a transition (4_{q_f}) can only be fired if $q_f \in Q_f$.
  - $v \notin \mathcal{L}(\mathcal{B})$: because no transition (4_{q_f}) could be fired, either $\mathcal{B}$ cannot read v or, after reading v, $\mathcal{B}$ is not in any state of Q'_f.

  Hence $v \in \mathcal{L}(\mathcal{A}) \setminus \mathcal{L}(\mathcal{B})$, proving that $\mathcal{L}(\mathcal{A}) \not\subseteq \mathcal{L}(\mathcal{B})$. □



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