From security protocols to pushdown automata

Rémy Chrétien, LORIA & LSV - CNRS, France
Véronique Cortier, LORIA - CNRS, France
Stéphanie Delaune, LSV, ENS Cachan & CNRS, France

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. We further show that checking for equivalence of protocols is actually equivalent to checking for equivalence of generalized, real-time deterministic pushdown automata.

Very recently, the algorithm for checking for equivalence of deterministic pushdown automata has been implemented. We have implemented our translation from protocols to pushdown automata, yielding the first tool that decides equivalence of (some class of) protocols, for an unbounded number of sessions. As an application, we have analyzed some protocols of the literature including a simplified version of the basic access control (BAC) protocol used in biometric passports.

Categories and Subject Descriptors: F.3.1 [**Logics and Meanings of Programs**]: Verifying and Reasoning about Programs

General Terms: Security

Additional Key Words and Phrases: formal proofs, security protocols, verification, trace equivalence

1. INTRODUCTION

Formal methods have been successfully applied for rigorously analyzing security protocols. In particular, many algorithms and tools (see [Rusinowitch and Turuani 2003; Blanchet 2001; Comon-Lundh and Cortier 2003; Basin et al. 2005; Cremers 2008] 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 indistinguishabil-

---

Author's address: S. Delaune, Laboratoire Spécification & Vérification - 61, avenue du président Wilson - 94 230 Cachan.
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.

DOI: 0000001.0000001

ity properties in [Arapinis et al. 2010; Bruso et al. 2010]). 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 in [Baudet 2005], it has been shown that trace equivalence is decidable for deterministic processes with no else branches, and for the family of convergent subterm equational theories [Cortier and Delaune 2009]. Convergent subterm theories capture most standard primitives including asymmetric and symmetric encryption, hashes, signatures, and macs. A simplified proof of [Baudet 2005] has been proposed by Y. Chevalier and M. Rusinowitch [Chevalier and Rusinowitch 2012].
— A. Tiu and J. Dawson [Tiu and Dawson 2010] 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 without else branches, and for symmetric encryption and pairs only.
— V. Cheval *et al.* [Cheval et al. 2011] have proposed and implemented a procedure for trace equivalence, and for a quite general class of processes that use standard primitives (symmetric and asymmetric encryption, hashes, signatures, pairs). In particular, this is the only decidability result that can consider non deterministic processes and 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 [Blanchet 2001], a well-established tool for checking security of protocols. ProVerif is able to check equivalence although it does not always succeed [Blanchet et al. 2005]. It can check equivalence of *bi-processes*, that is of two processes that have the same structure. ProVerif has been recently extended [Cheval and Blanchet 2013] to handle more processes, in particular with else branches but can still not consider processes with very different structures. 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. In the past, several decidable fragments and semi-decision procedures have been proposed for secrecy and authentication, for an unbounded number of sessions. Our goal is to obtain analogous results in the case of equivalence properties. We therefore focus on a class of protocols for which secrecy is decidable [Comon-Lundh and Cortier 2003]. This class, called *ping-pong protocols*, typically assumes that each protocol rule manipulates at most one variable and that the protocol is formed of a set of independent in/out rules. Intuitively, this corresponds to the assumption that, at each step of the protocol, upon receiving a message there is at most one part of it that is unknown to the agent (typically a key, a nonce, or an encrypted packet).

Surprisingly, while this class is decidable for reachability, even a fragment of it (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 and asymmetric encryption, and signatures. Since we need to assume our constructors to be randomized and since we assume "at most one variable", we can only handle a very limited notion of (randomized) concatenation that appends atomic values.

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 [Sénizergues 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:

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

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

To transform equivalence of processes into equivalence of pushdown automata, we first show how to get rid of an active attacker. More precisely, we show that

$$P \approx Q \Leftrightarrow P' \approx_{\mathsf{fwd}} Q'$$

where $\approx_{\mathsf{fwd}}$ intuitively represents equivalence of processes when the attacker may only *forward* messages. This equivalence is obtained by partially encoding the attacker in P' and Q', still preserving equivalence.

The decision procedure for checking equivalence of deterministic pushdown automata has been recently implemented by G. Sénizergues [Henry and Sénizergues 2013]. We have therefore implemented our tranformation from processes to pushdown automata, yielding the first tool that decides equivalence of (some class of) protocols for an unbounded number of sessions. As an application, we have analyzed several protocols of the literature, including a simplified version of the basic access control protocol (BAC) of the biometric passport [ICAO 2008].

We introduce the process algebra and its semantics in Section 2. We characterize the notion of ping-pong protocols in Section 3 and state our main results. Sections 4 and 5 are devoted to decidability. More precisely, we show in Section 4 how to get rid of an active attacker by encoding it directly in the process. Next, we show in Section 5 how to encode equivalence between processes (in presence of a forwarder attacker) into equivalence of pushdown automata, characterizing further which cases may result in non equivalence. Finally, we study in Section 6 the converse translation and show that equivalence of pushdown automata can be reduced to equivalence of protocols. We present our implementation and its application to protocols in Section 7. Concluding remarks can be found in Section 8.

2. MODEL FOR SECURITY PROTOCOLS

Security protocols are modeled through a process algebra that manipulates terms. We first give the syntax of our calculus in Section 2.1, before describing its semantics in Section 2.2. Then, in Section 2.3, we define the notion of equivalence of processes.

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, SymKey, 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 SymKey, 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:

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

We discuss in Section 7 how we can handle a limited notion of (randomized) concatenation.

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

$$\mathsf{sdec}(\mathsf{senc}(x, \mathsf{k}_1, z), \mathsf{k}_1) \to x$$

$$\mathsf{adec}(\mathsf{aenc}(x, \mathsf{k}_2, z), {\mathsf{k}_2}^{-1}) \to x \qquad \mathsf{check}(\mathsf{sign}(x, \mathsf{k}_3, z), {\mathsf{k}_3}^{-1}) \to x$$

with k_1 of sort SymKey, k_2 of sort PubKey, and k_3 of sort PrivKey. For instance, the first rule models the fact that the decryption of a ciphertext will return the associated plaintext when the right key is used to perform decryption. The two last rules are used to model asymmetric encryption and signatures. We denote by $t{\downarrow}$ the *normal form* of a term $t \in \mathcal{T}(\Sigma^+, \mathcal{N}, \mathcal{X})$.

Example 2.1. 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:

$P, Q :=$	0	null process
	$\mid\ \mathsf{in}(c, u).P$	input
	$\mid\ \mathsf{out}(c, u).P$	output
	$\mid\ (P \mid Q)$	parallel
	$\mid\ !P$	replication
	$\mid\ \mathsf{new}\ n.P$	name generation

where $u \in \mathcal{T}(\Sigma, \mathcal{N}, \mathcal{X})$, $n \in \mathcal{N}$, and $c \in \mathcal{C}h$.

The process 0 does nothing, and we sometimes omit it. 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 are 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.

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 2.2. We consider a simplified version of the protocol presented in [Denning and Sacco 1981]. The purpose of this protocol informally described below is to establish a key k_{AB} betwen two participants A and B using public key encryption and signature.

$$
\begin{array}{ll}
1. & A \to B:\ \mathsf{aenc}(\mathsf{sign}(\mathsf{k}_{AB}, \mathsf{sk}_A, r_A^1), \mathsf{pk}_B, r_A^2) \\
2. & B \to A:\ \mathsf{ack}
\end{array}
$$

The agent A sends a symmetric key k_{AB} signed with A's private key sk_A (using a fresh random number r_A^1), and the resulting ciphertext is encrypted with B's public key pk_B (using a fresh random number r_A^2). The agent B answers to this request by decrypting this message, and verifying the signature. If all checks succeed, B informs the agent A by sending an acknowledgement, *i.e.* the constant ack. The agents A and B can now use the symmetric key k_{AB} to communicate.

The role of A is modeled by a process P_A while the role of B is modeled by P_B. We have that:

$$
\begin{array}{rll}
P_A \stackrel{\mathsf{def}}{=} & !\,\mathsf{in}(c_A, \mathsf{start}).\mathsf{new}\, r_A^1.\mathsf{new}\, r_A^2.\mathsf{out}(c_A, \mathsf{aenc}(\mathsf{sign}(\mathsf{k}_{AB}, \mathsf{sk}_A, r_A^1), \mathsf{pk}_B, r_A^2)) & (1) \\
& \mid\ !\,\mathsf{in}(c'_A, \mathsf{start}).\mathsf{new}\, r_A^1.\mathsf{new}\, r_A^2.\mathsf{out}(c'_A, \mathsf{aenc}(\mathsf{sign}(\mathsf{k}_{AC}, \mathsf{sk}_A, r_A^1), \mathsf{pk}_C, r_A^2)) & (2) \\
P_B \stackrel{\mathsf{def}}{=} & !\,\mathsf{in}(c_B, \mathsf{aenc}(\mathsf{sign}(x, \mathsf{sk}_A, z_1), \mathsf{pk}_B, z_2)).\mathsf{out}(c_B, \mathsf{ack}) & (3)
\end{array}
$$

The constants c_A, c'_A and c_B are constants of sort $\mathsf{channel}$, ack is a constant of sort msg, whereas the constants $\mathsf{k}_{AB}, \mathsf{k}_{AC}, \mathsf{sk}_A, \mathsf{sk}_B, \mathsf{sk}_C, \mathsf{pk}_A, \mathsf{pk}_B, \mathsf{pk}_C$ which are in Σ_0 are such that:

— $\mathsf{k}_{AB}, \mathsf{k}_{AC}$ are of sort SymKey,
— $\mathsf{sk}_A, \mathsf{sk}_B, \mathsf{sk}_C$ are of sort PrivKey, and
— $\mathsf{pk}_A, \mathsf{pk}_B, \mathsf{pk}_C$ of sort PubKey.

Moreover, we have that $\mathsf{sk}_X^{-1} = \mathsf{pk}_X$ for $X \in \{A, B, C\}$ whereas $\mathsf{k}_{AB}^{-1} = \mathsf{k}_{AB}$ and $\mathsf{k}_{AC}^{-1} = \mathsf{k}_{AC}$. Finally, r_A^1, r_A^2 are names of sort rand, and x (resp. z_1, z_2) is a variable of sort msg (resp. rand).

Intuitively, P_A sends k_{AB} signed with sk_A and encrypted with pk_B to the agent B (branch 1). More generally, the agent A can start different sessions with different agents. Thus, the process P_A models the agent A initiating a session with B (branch 1) as well as with C (branch 2). The process P_B models the agent B answering a request from A. We could also consider the scenario where the agent B is also willing to talk to C or where the initiator, here played by A, is also played by other agents such as B. We consider here only a simpler case to keep the example reasonably short.

To model the whole protocol, we sent the public key $\mathsf{pk}_A, \mathsf{pk}_B, \mathsf{pk}_C$ in clear, as well as the private key sk_C, to model the fact that the attacker may learn the private keys of some corrupted agents. This is modeled through the following process P_{key}:

$$\begin{array}{rl} P_{\mathsf{key}} \stackrel{\mathsf{def}}{=} & !\,\mathsf{in}(c_1, \mathsf{start}).\mathsf{out}(c, \mathsf{pk}_A) \;\; | \, !\,\mathsf{in}(c_2, \mathsf{start}).\mathsf{out}(c, \mathsf{pk}_B) \;\; | \\ & !\,\mathsf{in}(c_3, \mathsf{start}).\mathsf{out}(c, \mathsf{pk}_C) \;\; | \, !\,\mathsf{in}(c_4, \mathsf{start}).\mathsf{out}(c, \mathsf{sk}_C) \end{array}$$

Then, the whole protocol is given by P, where P_A, P_B, and P_{key} evolve in parallel:

$$P \stackrel{\mathsf{def}}{=} P_A \;\; | \;\; P_B \;\; | \;\; P_{\mathsf{key}}$$

This protocol is actually insecure as demonstrated by the following attack:

$$\begin{array}{lll} 1. & A \to C: & \mathsf{aenc}(\mathsf{sign}(\mathsf{k}_{AC}, \mathsf{sk}_A, r_A^1), \mathsf{pk}_C, r_A^2) \\ 2. & C(A) \to B: & \mathsf{aenc}(\mathsf{sign}(\mathsf{k}_{AC}, \mathsf{sk}_A, r_A^1), \mathsf{pk}_B, r_C^1) \\ 3. & B \to A: & \mathsf{ack} \end{array}$$

A initiates a session with a malicious user C sending him a key k_{AC}. This malicious user then legally learns k_{AC} but also its signature $\mathsf{sign}(\mathsf{k}_{AC}, \mathsf{sk}_A, r_A^1)$ under the signing key of A. He may then resend this key to B in the name of A. The agent B accepts the key k_{AC} as being a secret key between A and B.

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 described in Figure 1. For sake of simplicity, we often write P instead of $(P; \emptyset)$.

The first rule (IN) allows the attacker to make a process progress by feeding it with a term he built with publicly available terms and symbols. The second one (OUT) lets the attacker gain knowledge of a message as soon as it is sent by a process: the corresponding message is added to the substitution of the current configuration. These two rules are the only observable actions. The two remaining rules are quite standard and are unobservable (τ action) from the point of view of the attacker.

The relation $\xrightarrow{\mathsf{tr}}$ between configurations (where tr is a sequence of actions) is defined in a usual way as the reflexive and transitive closure of the relation $\xrightarrow{\alpha}$. Given a se-

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

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

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

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

Fig. 1. Operational semantics.

quence of observable actions tr, we write $K \stackrel{\mathsf{tr}}{\Longrightarrow} K'$ when there exists tr' such that $K \xrightarrow{\mathsf{tr}'} K'$ and w is obtained from tr' 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 2.3. Going back to the protocol introduced in Example 2.2, we consider the scenario corresponding to the attack.

(1) The public keys of all the participants are disclosed as well as the secret key sk_C of the corrupted agent C. Formally, let $K_0 \stackrel{\mathsf{def}}{=} (P;\emptyset)$, we have that:

$$K_0 \xrightarrow{\mathsf{in}(c_1,\mathsf{start}).\mathsf{out}(c_1,\mathsf{w}_1).\mathsf{in}(c_2,\mathsf{start}).\mathsf{out}(c_2,\mathsf{w}_2).\mathsf{in}(c_3,\mathsf{start}).\mathsf{out}(c_3,\mathsf{w}_3)} \xrightarrow{\mathsf{in}(c_4,\mathsf{start}).\mathsf{out}(c_4,\mathsf{w}_4)} (P;\sigma_0)$$

where $\sigma_0 = \{\mathsf{w}_1 \rhd \mathsf{pk}_A,\ \mathsf{w}_2 \rhd \mathsf{pk}_B,\ \mathsf{w}_3 \rhd \mathsf{pk}_C,\ \mathsf{w}_4 \rhd \mathsf{sk}_C\}$.

(2) The agent A initiates a session with C and sends the corresponding encrypted message. More formally, we have that:

$$(P;\sigma_0) \xrightarrow{\mathsf{in}(c'_A,\mathsf{start}).\mathsf{out}(c'_A,\mathsf{w}_5)} (P;\sigma)$$

where $\sigma = \sigma_0 \cup \{\mathsf{w}_5 \rhd \mathsf{aenc}(\mathsf{sign}(\mathsf{k}_{AC},\mathsf{sk}_A,r^1_A),\mathsf{pk}_C,r^2_A)\}$ and r^1_A, r^2_A are (fresh) names in $\mathcal{N}_{\mathsf{prv}}$.

Hence, we have that $(\mathsf{tr},\sigma) \in \mathsf{trace}(K_0)$ where:

$$\begin{aligned}\mathsf{tr} = \ & \mathsf{in}(c_1,\mathsf{start}).\mathsf{out}(c_1,\mathsf{w}_1).\mathsf{in}(c_2,\mathsf{start}).\mathsf{out}(c_2,\mathsf{w}_2).\mathsf{in}(c_3,\mathsf{start}).\mathsf{out}(c_3,\mathsf{w}_3).\\ & \mathsf{in}(c_4,\mathsf{start}).\mathsf{out}(c_4,\mathsf{w}_4).\mathsf{in}(c'_A,\mathsf{start}).\mathsf{out}(c'_A,\mathsf{w}_5).\end{aligned}$$

In this execution trace, first the keys $\mathsf{pk}_A, \mathsf{pk}_B, \mathsf{pk}_C$ and sk_C are sent after having called the corresponding process. Then, branch (2) of P is 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 [Arapinis et al. 2010; Bruso et al. 2010]. 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 2.4. 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 2.5. Consider the two following frames:

(1) $\sigma_1 \overset{\mathsf{def}}{=} \sigma_0 \cup \{\mathsf{w}_5 \triangleright \mathsf{aenc}(\mathsf{sign}(\mathsf{k}_{AC}, \mathsf{sk}_A, r_A^1), \mathsf{pk}_C, r_A^2),\ \mathsf{w}_6 \triangleright \mathsf{k}_{AC}\}$,
(2) $\sigma_2 \overset{\mathsf{def}}{=} \sigma_0 \cup \{\mathsf{w}_5 \triangleright \mathsf{aenc}(\mathsf{sign}(\mathsf{k}_{AC}, \mathsf{sk}_A, r_A^1), \mathsf{pk}_C, r_A^2),\ \mathsf{w}_6 \triangleright \mathsf{k}\}$.

where k is a (private) constant in Σ_0. We have that $\sigma_1 \not\sim \sigma_2$. Indeed, consider the recipes $R_1 = \mathsf{check}(\mathsf{adec}(\mathsf{w}_5, \mathsf{w}_4), \mathsf{w}_1)$ and $R_2 = \mathsf{w}_6$. We have that $R_1\sigma_1\!\downarrow = R_2\sigma_1\!\downarrow = \mathsf{k}_{AC}$, whereas $R_1\sigma_2\!\downarrow = \mathsf{k}_{AC}$ and $R_2\sigma_2\!\downarrow = \mathsf{k}$ thus $R_1\sigma_2\!\downarrow \neq R_2\sigma_2\!\downarrow$.

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

Definition 2.6. 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 2.7. Continuing Example 2.2, our naive protocol is secure if the key received by B remains private. To model this, we modify the process P_B as follows:

$$P_B^l \overset{\mathsf{def}}{=} \ !\,\mathsf{in}(c_B, \mathsf{aenc}(\mathsf{sign}(x, \mathsf{sk}_A, z_1), \mathsf{pk}_B, z_2)).\mathsf{out}(c_B, x)$$
$$P_B^r \overset{\mathsf{def}}{=} \ !\,\mathsf{in}(c_B, \mathsf{aenc}(\mathsf{sign}(x, \mathsf{sk}_A, z_1), \mathsf{pk}_B, z_2)).\mathsf{out}(c_B, \mathsf{k})$$

Then, to model secrecy of the key received by B, we consider the following equivalence: $P_A \mid P_B^l \mid P_{\mathsf{key}} \approx P_A \mid P_B^r \mid P_{\mathsf{key}}$. An attacker should not distinguish between two instances of the protocol, one where B used the key established through the protocol and one where a magic key k is used instead.

However, our protocol is insecure. An attacker may easily learn k_{AC}, and sends to B a message of the expected form (as if it was issued by A) and that will contain this corrupted key instead of k_{AB}. Formally, we have that:

$$P_A \mid P_B^l \mid P_{\mathsf{key}} \not\approx P_A \mid P_B^r \mid P_{\mathsf{key}}.$$

This is reflected by the trace tr' described below:

$$\mathsf{tr}' \overset{\mathsf{def}}{=} \mathsf{tr}.\mathsf{in}(c_B, \mathsf{aenc}(\mathsf{adec}(\mathsf{w}_5, \mathsf{sk}_C), \mathsf{w}_2, r_C)).\mathsf{out}(c_B, \mathsf{w}_6)$$

where r_C is a name in $\mathcal{N}_{\mathsf{pub}}$.

We have that $(\mathsf{tr}', \sigma_1) \in \mathsf{trace}(K_0)$ with $K_0 = (P_A \mid P_B^l \mid P_{\mathsf{key}};\, \sigma_1)$ and σ_1 as defined in Example 2.5. Because of the existence of only one branch using each channel, there is only one possible execution of $P_A \mid P_B^r \mid P_{\mathsf{key}}$ (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_2 as described in Example 2.5. We have already seen that static equivalence does not hold, *i.e.* $\sigma_1 \not\sim \sigma_2$.

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 [Comon-Lundh and Cortier 2003].

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 $r \in \mathcal{N}$. Intuitively, no destructor should be used explicitly. Moreover, we assume that each variable (resp. name) occurs at most once in u (resp. v).

Definition 3.1. $\mathcal{C}_{\mathsf{pp}}$ is the class of protocol of the form:

$$P = \mathop{|}_{i=1}^{n} \mathop{|}_{j=1}^{p_i} !\mathsf{in}(c_i, u_i^j).\mathsf{new}\ r_1.\ldots.\mathsf{new}\ r_{k_j^i}.\,\mathsf{out}(c_i, v_i^j)$$

such that:

(1) for all $i \in \{1, \ldots, n\}$, and $j \in \{1, \ldots, p_i\}$, $k_i^j \in \mathbb{N}$, u_i^j is an input term, and v_i^j is an output term where names occurring in v_i^j are included in $\{r_1, \ldots, r_{k_i^j}\}$;

(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, $u_i^{j_1}$ and $u_i^{j_2}$ are not unifiable[1].

Each subprocess $\mathsf{in}(c_i, u_i^j).\mathsf{new}\ r_1.\ldots.\mathsf{new}\ r_{k_j^i}.\,\mathsf{out}(c_i, v_i^j)$ is called a *branch* of P.

Item *1* holds for any process representing a protocol: the variables of the output should be bound by the input. Item *2* enforces 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 $\mathcal{C}_{\mathsf{pp}}$ is that we consider a restricted signature (e.g. no pair, no hash function), and names can only be used to produce randomized ciphertexts.

Example 3.2. The protocols described in Example 2.7 are in $\mathcal{C}_{\mathsf{pp}}$. For instance, we can check that:

— $\mathsf{aenc}(\mathsf{sign}(x, \mathsf{sk}_A, z_1), \mathsf{pk}_B, z_2)$ is an input term, and
— $\mathsf{aenc}(\mathsf{sign}(\mathsf{k}_{AB}, \mathsf{sk}_A, r_A^1), \mathsf{pk}_B, r_A^2)$ is an output term.

Moreover, the determinism condition (item *2*) is clearly satisfied: each branch of the protocol $P_A \mid P_B^l \mid P_{\mathsf{key}}$ (resp. $P_A \mid P_B^r \mid P_{\mathsf{key}}$) uses a different channel.

When studying trace equivalence (or even trace inclusion) we can even safely force a process to perform an input action followed directly by its associated output action.

We consider a set of "big-step" traces, defined as follows.

$$\mathsf{trace}^{\mathsf{io}*}(K) \;=\; \left\{ (\mathsf{tr}, \sigma) \;\middle|\; \begin{array}{c} K \xRightarrow{\mathsf{tr}} (\mathcal{P}; \sigma) \text{ for some configuration } (\mathcal{P}; \sigma) \\ \text{with } \mathsf{tr} \text{ sequence of input-output blocks.} \end{array} \right\}$$

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

The notion of trace inclusion (resp. trace equivalence) w.r.t. big-step traces is defined accordingly.

Definition 3.3. Let P and Q be two protocols. We have that $P \sqsubseteq^{\mathsf{io}*} Q$ if for every $(\mathsf{tr}, \sigma) \in \mathsf{trace}^{\mathsf{io}*}(P)$, there exists $(\mathsf{tr}', \sigma') \in \mathsf{trace}^{\mathsf{io}*}(Q)$ such that $\mathsf{tr} = \mathsf{tr}'$ and $\sigma \sim \sigma'$. They are *trace equivalent*, written $P \approx^{\mathsf{io}*} Q$, if $P \sqsubseteq^{\mathsf{io}*} Q$ and $Q \sqsubseteq^{\mathsf{io}*} P$.

Due to the form of protocols in $\mathcal{C}_{\mathsf{pp}}$, any trace made up of inputs and outputs actions can be first completed with all the available output actions, and then be mapped to a trace that is made up of input-ouput blocks only. Thus, we have that the two notions of trace equivalence coincide.

PROPOSITION 3.4. *Let P and Q be two protocols in $\mathcal{C}_{\mathsf{pp}}$. We have that $P \sqsubseteq^{\mathsf{io}*} Q$ if, and only if, $P \sqsubseteq Q$.*

This proposition easily follows from that fact that for any process of $\mathcal{C}_{\mathsf{pp}}$, any input is immediately followed by an output.

3.2. Main results

Our first main contribution is a decision procedure for trace equivalence of processes in $\mathcal{C}_{\mathsf{pp}}$.

THEOREM 3.5. *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.*

Deciding trace equivalence is done in two main steps.

(1) First, we show how to reduce trace equivalence 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*, that is, when the attacker may only forward messages obtained through the protocol. This step is detailed in Section 4.
(2) 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), that is, deterministic pushdown automata with no epsilon-transition but such that the automata may unstack several symbols at a time. This step is detailed in Section 5

We also provide an implementation of our translation from protocols to pushdown automata, yielding a tool for automatically checking equivalence of security protocols, for an unbounded number of sessions. This contribution is described in Section 7.

Actually, we characterize equivalence of protocols in terms of equivalence of GPDA. Indeed, Step (2) above shows 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. Conversely, we also 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. This reverse encoding, from pushdown automata to protocols is explained in Section 6.

Our second contribution is an undecidability result. The class $\mathcal{C}_{\mathsf{pp}}$ is somewhat limited but extending $\mathcal{C}_{\mathsf{pp}}$ to non deterministic processes immediately yields undecidability of trace equivalence. More precisely, we have that trace inclusion of processes in $\mathcal{C}_{\mathsf{pp}}$ is undecidable.

THEOREM 3.6. *The following problem is undecidable.*

Input. *P and Q two protocols in $\mathcal{C}_{\mathsf{pp}}$.*

Output. *Whether P is trace included in Q, i.e. $P \sqsubseteq Q$.*

A direct encoding of the Post Correspondance Problem (PCP) into an inclusion of two protocols of this class is given in Appendix A. Alternatively, this undecidability result is also a consequence of the reduction result established in Section 6 and the undecidability result established in [Friedman 1976]. Nonetheless, we present in Appendix A the direct encoding of PCP into protocol equivalence since some ideas might be reused to show undecidability of trace equivalence for some other classes whereas the alternative proof required a first encoding to transform a protocol into a pushdown automaton.

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 3.7. *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. GETTING RID OF THE FULL ATTACKER

We show in this section how to reduce trace equivalence 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*, that is, when the attacker may only forward messages obtained through the protocols. This new semantics induced a new notion of trace equivalence, denoted $\approx_{\mathsf{fwd}}$, which is formally defined in Section 4.1.

To counterbalance the effects of this simple forwarder 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 $\mathcal{T}_{\mathsf{fwd}}$ (see Section 4.2) 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 4.1. *Let P and Q be two protocols in $\mathcal{C}_{\mathsf{pp}}$. We have that:*

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

4.1. Forwarder semantics

We first define the actions of a forwarder by modifying our semantics. Roughly, we restrict the recipes R, R_1, and R_2 that are used in the IN rule and in static equivalence (Definition 2.4) to be either the public constant start or a variable in $\mathcal{W}$. Intuitively, this corresponds to the fact that the forwarder attacker should no longer build a message on his own. This leads us to consider a new relation $\rightarrow_{\mathsf{fwd}}$ between configurations which is the relation induced by the rules described in Figure 2.

The relations $\xrightarrow{\mathsf{tr}}_{\mathsf{fwd}}$ and $\xRightarrow{\mathsf{tr}}_{\mathsf{fwd}}$ between configurations where tr is a sequence of actions (resp. observable actions) are defined as expected. For every configuration K, we define its *set of traces w.r.t. the forwarder semantics* as follows:

$$\mathsf{trace}_{\mathsf{fwd}}(K) = \left\{ (\mathsf{tr}, \sigma) \;\middle|\; \begin{array}{l} K \xRightarrow{\mathsf{tr}}_{\mathsf{fwd}} (\mathcal{P}; \sigma) \text{ for some configuration } (\mathcal{P}; \sigma) \\ \text{with } \mathsf{tr} \text{ sequence of input-output blocks.} \end{array} \right\}$$

We need also to adapt our notion of static equivalence.

$$(\mathsf{in}(c,u).P \cup \mathcal{P}; \sigma) \xrightarrow{\mathsf{in}(c,R)}_{\mathsf{fwd}} (P\theta \cup \mathcal{P}; \sigma) \quad \text{where } R \in \{\mathsf{start}\} \cup \mathcal{W} \text{ and } R\sigma\!\downarrow = u\theta \text{ for some } \theta$$

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

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

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

Fig. 2. Semantics for a forwarder attacker.

Definition 4.2. Two frames σ_1 and σ_2 are *statically equivalent w.r.t. the forwarder semantics*, denoted $\sigma_1 \sim_{\mathsf{fwd}} \sigma_2$, when we have that $dom(\sigma_1) = dom(\sigma_2)$, and for all recipes R_1 and R_2 in $\{\mathsf{start}\} \cup \mathcal{W}$, we have that $R_1\sigma_1 = R_2\sigma_1$ if, and only if, $R_1\sigma_2 = R_2\sigma_2$.

This induces a new notion of trace equivalence which is formally defined as follows:

Definition 4.3. Let P and Q be two protocols. We have that $P \sqsubseteq_{\mathsf{fwd}} Q$ if for every $(\mathsf{tr}, \sigma) \in \mathsf{trace}_{\mathsf{fwd}}(P)$, there exists $(\mathsf{tr}', \sigma') \in \mathsf{trace}_{\mathsf{fwd}}(Q)$ such that $\mathsf{tr} = \mathsf{tr}'$ and $\sigma \sim_{\mathsf{fwd}} \sigma'$. They are *trace equivalent w.r.t. the forwarder semantics*, written $P \approx_{\mathsf{fwd}} Q$, if $P \sqsubseteq_{\mathsf{fwd}} Q$ and $Q \sqsubseteq_{\mathsf{fwd}} P$.

Example 4.4. The trace exhibited in Example 2.3 is still a valid one according to the forwarder semantics, but the frames σ_1 and σ_2 described in Example 2.5 are now in equivalence w.r.t. $\sim_{\mathsf{fwd}}$. Actually, we have that $P_A \mid P_B^l \mid P_{\mathsf{key}} \approx_{\mathsf{fwd}} P_A \mid P_B^r \mid P_{\mathsf{key}}$. Indeed, the fact that a forwarder simply acts as a relay prevents him to mount the aforementioned attack.

4.2. Towards a forwarder attacker

As illustrated in Example 4.4, the forwarder semantics is very restrictive: a forwarder cannot rely on his deduction capabilities to mount an attack. We show however that we can still restrict ourselves to trace equivalence w.r.t. a forwarder.

Intuitively, we transform any two processes P, Q into processes $\bar{P}, \bar{Q}$ such that $P \approx Q$ if and only if $\bar{P} \approx_{\mathsf{fwd}} \bar{Q}$. Roughly this transformation consists in two steps.

(1) First, we guess 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 $P \approx Q$.
(2) 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 branches in the processes.

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

Example 4.5. To better illustrate this section, we consider a variant of the processes introduced in Section 2., where agent A is now willing to talk only to B.

$$P \stackrel{\mathsf{def}}{=} P'_A \mid P_B^l \mid P_{\mathsf{key}} \qquad Q \stackrel{\mathsf{def}}{=} P'_A \mid P_B^r \mid P_{\mathsf{key}}$$

where P_B^l, P_B^r are defined in Example 2.7 and P_{key} is defined in Example 2.2, whereas P'_A is defined as follows (only the first branch of P_A)

$$P'_A \stackrel{\mathsf{def}}{=} \; !\,\mathsf{in}(c_A, \mathsf{start}).\mathsf{new}\, r_A^1.\mathsf{new}\, r_A^2.\mathsf{out}(c_A, \mathsf{aenc}(\mathsf{sign}(\mathsf{k}_{AB}, \mathsf{sk}_A, r_A^1), \mathsf{pk}_B, r_A^2)) \quad (1)$$

This scenario excludes the aforementioned attack and we have that $P \approx Q$. This has been formally checked using our prototype (see Section 7).

4.2.1. Guessing deducible keys. The purpose of this section is to restrict our attention to protocols that explicitly disclose their deducible keys $\mathcal{K}_P$ and $\mathcal{K}_Q$. Since we do not want to rely on a particular procedure for computing these two sets, the idea is to guess a possibly superset of each set, namely K and K', and then ensure that these sets K and K' contain *at least* the deducible keys.

Definition 4.6. Let P be a protocol in $\mathcal{C}_{\mathsf{pp}}$. A term t is *deducible* in P if there exists a trace $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ and a recipe R (*i.e.* a term in $\mathcal{T}(\Sigma_{\mathsf{pub}}, \mathcal{N}_{\mathsf{pub}}, \mathcal{W})$) such that $R\phi\!\downarrow = t$.

Example 4.7. Continuing Example 4.5, we have that P and Q are in $\mathcal{C}_{\mathsf{pp}}$. It is easy to notice that k_{AB} is deducible in P whereas k is deducible in Q since these keys are revealed at the end of B's execution. For both P and Q, the trace $\mathsf{tr} = \mathsf{in}(c_A, \mathsf{start}).\mathsf{out}(c_A, \mathsf{w}_1).\mathsf{in}(c_B, \mathsf{w}_1).\mathsf{out}(c_B, w_2)$ and the recipe $R = \mathsf{w}_2$ is a witness of this fact.

Two equivalent processes have the same set of deducible keys, up to some bijective renaming.

LEMMA 4.8. *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 $\mathsf{s} \in \{\mathsf{SymKey}, \mathsf{PubKey}, \mathsf{PrivKey}\}$.
— *$R\phi\!\downarrow = \mathsf{k}$ if, and only if, $R\psi\!\downarrow = \alpha(\mathsf{k})$;*
— *$R\phi\!\downarrow = \mathsf{k}^{-1}$ if, and only if, $R\psi\!\downarrow = (\alpha(\mathsf{k}))^{-1}$;*

and conversely, for every $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$ there exists a trace $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ satisfying the same properties.

PROOF. *(sketch)* The relation α is defined as follows:

> for every $\mathsf{k} \in \mathcal{K}_P$ of sort s, and every trace $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ and recipe R such that $R\phi\!\downarrow = \mathsf{k}$, we define $\alpha(\mathsf{k}) = R\psi\!\downarrow$ where ψ is the only frame such that $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$.

The existence of such a frame comes from the fact that $P \approx Q$, whereas its unicity is a consequence of the determinism of protocols in $\mathcal{C}_{\mathsf{pp}}$.

Then, we show that this relation α is uniquely defined and satisfied all the requirements exploiting the strong relationship between P and Q through the relation $P \approx Q$. □

Example 4.9. Continuing Example 4.5, we have $\mathcal{K}_P = \{\mathsf{pk}_A, \mathsf{pk}_B, \mathsf{pk}_C, \mathsf{sk}_C, \mathsf{k}_{AB}\}$ whereas $\mathcal{K}_Q = \{\mathsf{pk}_A, \mathsf{pk}_B, \mathsf{pk}_C, \mathsf{sk}_C, \mathsf{k}\}$. The unique bijection α mentioned in the previous lemma is defined as follows: $\alpha(\mathsf{k}_{AB}) = \mathsf{k}$, and $\alpha(\mathsf{k}') = \mathsf{k}'$ otherwise.

Definition 4.10. 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*.

Example 4.11. Continuing our running example, P and Q clearly disclose $K = \{\mathsf{pk}_A, \mathsf{pk}_B, \mathsf{pk}_C, \mathsf{sk}_C\}$.

LEMMA 4.12. *Let P and Q be two protocols in $\mathcal{C}_{\mathsf{pp}}$, S (resp. S') the set of keys of P (resp. Q). Then $P \approx Q$ 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:*

$$\begin{aligned}\bar{P} = P \;\mid\; & !\mathsf{in}(c^0, \mathsf{start}).\mathsf{out}(c^0, 0) | !\mathsf{in}(c^1, \mathsf{start}).\mathsf{out}(c^1, 1)\\ \mid\; & \big|_{\mathsf{k}\in K} !\mathsf{in}(c_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{start}).\mathsf{out}(c_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{k}) \big|_{\mathsf{k}\in S\smallsetminus K} !\mathsf{in}(c, \mathsf{k}).\mathsf{out}(c, 0)\\ \bar{Q} = Q \;\mid\; & !\mathsf{in}(c^0, \mathsf{start}).\mathsf{out}(c^0, 0) | !\mathsf{in}(c^1, \mathsf{start}).\mathsf{out}(c^1, 1)\\ \mid\; & \big|_{\mathsf{k}\in K} !\mathsf{in}(c_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{start}).\mathsf{out}(c_{\mathsf{k},\alpha(\mathsf{k})}, \alpha(\mathsf{k})) \big|_{\mathsf{k}\in S'\smallsetminus K'} !\mathsf{in}(c, \mathsf{k}).\mathsf{out}(c, 1)\end{aligned}$$

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 $\mathcal{T}_{\mathsf{key}}(P, Q)$ the set of such pairs $(\bar{P}, \bar{Q})$ of modified protocols.

PROOF. Let $\mathcal{K}_P$ (resp. $\mathcal{K}_Q$) be the set of deducible constants of sort key that occur in P (resp. Q). We prove the two directions separately.

($\Rightarrow$) If $P \approx Q$, by Lemma 4.8, for $K = \mathcal{K}_P$ and $K' = \mathcal{K}_Q$, we get the existence of such a bijection α. Because keys in $S \smallsetminus \mathcal{K}_P$ and $S' \smallsetminus \mathcal{K}_Q$ are not deducible, the branches on channel c can never be triggered. Moreover, 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). Indeed, for every couple $(\mathsf{k}, \mathsf{k}^{-1})$ of deducible keys and for any recipe reducing to k (resp. k^{-1}) in P, the same recipe reduces to $\alpha(\mathsf{k})$ (resp. $\alpha(\mathsf{k})^{-1}$) in Q, thanks to the properties of α described in Lemma 4.8.

($\Leftarrow$) For the converse implication, we first remark that necessarily we have that $\mathcal{K}_P \subseteq K$ and $\mathcal{K}_Q \subseteq K'$. Indeed, suppose there exists, for instance, $\mathsf{k} \in \mathcal{K}_P \smallsetminus K$. Since k is deducible, there exists a trace $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ and a recipe R such that $R\phi\!\downarrow = \mathsf{k}$. Since (tr, ϕ) is also a trace of $\bar{P}$, we consider the trace:

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

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. We conclude in a similar way in case $\mathsf{k} \in \mathcal{K}_Q \smallsetminus K'$.

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. □

Example 4.13. Continuing our example, let $K = \mathcal{K}_P$ and $K' = \mathcal{K}_Q$, and α the bijection defined in Example 4.9. Checking equivalence of $P \approx Q$ amounts into checking whether $\bar{P} \approx \bar{Q}$ where $\bar{P}$ and $\bar{Q}$ are defined as follows.

$$\begin{aligned}\bar{P} = P \;\mid\; & !\mathsf{in}(c^0, \mathsf{start}).\mathsf{out}(c^0, 0) | !\mathsf{in}(c^1, \mathsf{start}).\mathsf{out}(c^1, 1)\\ \mid\; & !\mathsf{in}(c_{\mathsf{k}_{AB},\mathsf{k}}, \mathsf{start}).\mathsf{out}(c_{\mathsf{k}_{AB},\mathsf{k}}, \mathsf{k}_{AB})\\ \mid\; & !\mathsf{in}(c, \mathsf{sk}_A).\mathsf{out}(c, 0) \;| !\mathsf{in}(c, \mathsf{sk}_B).\mathsf{out}(c, 0)\end{aligned}$$

$$\begin{aligned}\bar{Q} = Q \ \mid\ & !\mathsf{in}(c^0, \mathsf{start}).\mathsf{out}(c^0, 0) | !\mathsf{in}(c^1, \mathsf{start}).\mathsf{out}(c^1, 1)\\ \mid\ & !\mathsf{in}(c_{\mathsf{k}_{AB},\mathsf{k}}, \mathsf{start}).\mathsf{out}(c_{\mathsf{k}_{AB},\mathsf{k}}, \mathsf{k})\\ \mid\ & !\mathsf{in}(c, \mathsf{sk}_A).\mathsf{out}(c, 1) \ | !\mathsf{in}(c, \mathsf{sk}_B).\mathsf{out}(c, 1)\end{aligned}$$

If $\bar{P} \approx \bar{Q}$, then sk_A and sk_B cannot be deducible thus $\bar{P}$ and $\bar{Q}$ disclose their set of deducible keys.

4.2.2. Adding oracles. 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 branches in the processes. We rely on the bijection α computed in the previous section to do this in a compatible way on both sides of the equivalence.

LEMMA 4.14. *Let P and Q be two protocols in $\mathcal{C}_{\mathsf{pp}}$ respectively disclosing two sets of keys K and K' as in Lemma 4.12. Then $P \approx Q$ if, and only if, $\bar{P} \approx_{\mathsf{fwd}} \bar{Q}$ where:*

$$\begin{aligned}\bar{P} = P \ \mid\ & \big|_{\mathsf{k}\in K^{\mathsf{SymKey}}} !\mathsf{in}(c^{\mathsf{senc}}_{\mathsf{k},\alpha(\mathsf{k})}, x).\mathsf{new}\ n.\mathsf{out}(c_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{senc}(x, \mathsf{k}, n))\\ \mid\ & \big|_{\mathsf{k}\in K^{\mathsf{SymKey}}} !\mathsf{in}(c^{\mathsf{sdec}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{senc}(x, \mathsf{k}, y)).\mathsf{out}(c^{\mathsf{sdec}}_{\mathsf{k},\alpha(\mathsf{k})}, x)\\ \mid\ & \big|_{\mathsf{k}\in K^{\mathsf{PubKey}}} !\mathsf{in}(c^{\mathsf{aenc}}_{\mathsf{k},\alpha(\mathsf{k})}, x).\mathsf{new}\ n.\mathsf{out}(c^{\mathsf{aenc}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{aenc}(x, \mathsf{k}, n))\\ \mid\ & \big|_{\mathsf{k}\in K^{\mathsf{PrivKey}}} !\mathsf{in}(c^{\mathsf{adec}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{aenc}(x, \mathsf{k}, y)).\mathsf{out}(c^{\mathsf{adec}}_{\mathsf{k},\alpha(\mathsf{k})}, x)\\ \mid\ & \big|_{\mathsf{k}\in K^{\mathsf{PrivKey}}} !\mathsf{in}(c^{\mathsf{sign}}_{\mathsf{k},\alpha(\mathsf{k})}, x).\mathsf{new}\ n.\mathsf{out}(c^{\mathsf{sign}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{sign}(x, \mathsf{k}, n))\\ \mid\ & \big|_{\mathsf{k}\in K^{\mathsf{PubKey}}} !\mathsf{in}(c^{\mathsf{check}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{sign}(x, \mathsf{k}, y)).\mathsf{out}(c^{\mathsf{check}}_{\mathsf{k},\alpha(\mathsf{k})}, x)\end{aligned}$$

$$\begin{aligned}\bar{Q} = Q \ \mid\ & \big|_{\mathsf{k}\in K^{\mathsf{SymKey}}} !\mathsf{in}(c^{\mathsf{senc}}_{\mathsf{k},\alpha(\mathsf{k})}, x).\mathsf{new}\ n.\mathsf{out}(c^{\mathsf{senc}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{senc}(x, \alpha(\mathsf{k}), n))\\ \mid\ & \big|_{\mathsf{k}\in K^{\mathsf{SymKey}}} !\mathsf{in}(c^{\mathsf{sdec}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{senc}(x, \alpha(\mathsf{k}), y)).\mathsf{out}(c^{\mathsf{sdec}}_{\mathsf{k},\alpha(\mathsf{k})}, x)\\ \mid\ & \big|_{\mathsf{k}\in K^{\mathsf{PubKey}}} !\mathsf{in}(c^{\mathsf{aenc}}_{\mathsf{k},\alpha(\mathsf{k})}, x).\mathsf{new}\ n.\mathsf{out}(c^{\mathsf{aenc}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{aenc}(x, \alpha(\mathsf{k}), n))\\ \mid\ & \big|_{\mathsf{k}\in K^{\mathsf{PrivKey}}} !\mathsf{in}(c^{\mathsf{adec}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{aenc}(x, \alpha(\mathsf{k}), y)).\mathsf{out}(c^{\mathsf{adec}}_{\mathsf{k},\alpha(\mathsf{k})}, x)\\ \mid\ & \big|_{\mathsf{k}\in K^{\mathsf{PrivKey}}} !\mathsf{in}(c^{\mathsf{sign}}_{\mathsf{k},\alpha(\mathsf{k})}, x).\mathsf{new}\ n.\mathsf{out}(c^{\mathsf{sign}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{sign}(x, \alpha(\mathsf{k}), n))\\ \mid\ & \big|_{\mathsf{k}\in K^{\mathsf{PubKey}}} !\mathsf{in}(c^{\mathsf{check}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{check}(x, \alpha(\mathsf{k}), y)).\mathsf{out}(c^{\mathsf{check}}_{\mathsf{k},\alpha(\mathsf{k})}, x)\end{aligned}$$

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

PROOF. *(sketch)* First, thanks to Lemma 4.12, we know that P, $\bar{P}$, Q and $\bar{Q}$ disclose all their deducible keys.

($\Rightarrow$) Given a witness of non-equivalence for $\bar{P} \approx_{\mathsf{fwd}} \bar{Q}$, it is quite easy to build a witness of non-equivalence for $P \not\approx Q$ replacing the use of the oracle by the corresponding attacker construction. This yields a witness of non-equivalence for $P \approx Q$.

($\Leftarrow$) This direction is actually more involved. The idea is to replace the use of an attacker construction, e.g. an encryption with a deducible key, by the corresponding oracle. However, the attacker has the ability to use the same random seed more than once whereas this is impossible when using the oracles to perform those computations. Thus, we first show that this additional ability does not give any power to the attacker. Then, we do the replacement as expected in order to conclude.

The full proof is provided in Appendix B.2. □

Example 4.15. Continuing our example, this last transformation will add 10 branches (2 per deducible key). For instance, regarding the key k_{AB}, the two following branches will be be added:

For process P:

$$\begin{array}{l} !\mathsf{in}(c^{\mathsf{senc}}_{\mathsf{k}_{AB},\mathsf{k}}, x).\mathsf{new}\ n.\mathsf{out}(c_{\mathsf{k}_{AB},\mathsf{k}}, \mathsf{senc}(x, \mathsf{k}_{AB}, n)) \\ \mid\ !\mathsf{in}(c^{\mathsf{sdec}}_{\mathsf{k}_{AB},\mathsf{k}}, \mathsf{senc}(x, \mathsf{k}_{AB}, y)).\mathsf{out}(c^{\mathsf{sdec}}_{\mathsf{k}_{AB},\mathsf{k}}, x) \end{array}$$

For process Q:

$$\begin{array}{l} !\mathsf{in}(c^{\mathsf{senc}}_{\mathsf{k}_{AB},\mathsf{k}}, x).\mathsf{new}\ n.\mathsf{out}(c_{\mathsf{k}_{AB},\mathsf{k}}, \mathsf{senc}(x, \mathsf{k}, n)) \\ \mid\ !\mathsf{in}(c^{\mathsf{sdec}}_{\mathsf{k}_{AB},\mathsf{k}}, \mathsf{senc}(x, \mathsf{k}, y)).\mathsf{out}(c^{\mathsf{sdec}}_{\mathsf{k}_{AB},\mathsf{k}}, x) \end{array}$$

Regarding the keys $\mathsf{pk}_A, \mathsf{pk}_B, \mathsf{pk}_C$ and sk_A, since $\alpha(\mathsf{k}') = \mathsf{k}'$ for each of these keys, we add the following branches on both sides:

$$\begin{array}{l} \mid\ !\mathsf{in}(c^{\mathsf{aenc}}_{\mathsf{pk}_A,pk_A}, x).\mathsf{new}\ n.\mathsf{out}(c^{\mathsf{aenc}}_{\mathsf{pk}_A,\mathsf{pk}_A}, \mathsf{aenc}(x, \mathsf{pk}_A, n)) \\ \mid\ !\mathsf{in}(c^{\mathsf{aenc}}_{\mathsf{pk}_B,pk_B}, x).\mathsf{new}\ n.\mathsf{out}(c^{\mathsf{aenc}}_{\mathsf{pk}_B,\mathsf{pk}_B}, \mathsf{aenc}(x, \mathsf{pk}_B, n)) \\ \mid\ !\mathsf{in}(c^{\mathsf{aenc}}_{\mathsf{pk}_C,pk_C}, x).\mathsf{new}\ n.\mathsf{out}(c^{\mathsf{aenc}}_{\mathsf{pk}_C,\mathsf{pk}_C}, \mathsf{aenc}(x, \mathsf{pk}_C, n)) \\ \mid\ !\mathsf{in}(c^{\mathsf{adec}}_{\mathsf{sk}_C,\mathsf{sk}_C}, \mathsf{aenc}(x, \mathsf{pk}_C, y)).\mathsf{out}(c^{\mathsf{adec}}_{\mathsf{sk}_C,\mathsf{sk}_C}, x) \\ \mid\ !\mathsf{in}(c^{\mathsf{sign}}_{\mathsf{sk}_C,\mathsf{sk}_C}, x).\mathsf{new}\ n.\mathsf{out}(c^{\mathsf{sign}}_{\mathsf{sk}_C,\mathsf{sk}_C}, \mathsf{sign}(x, \mathsf{sk}_C, n)) \\ \mid\ !\mathsf{in}(c^{\mathsf{check}}_{\mathsf{pk}_A,\mathsf{pk}_A}, \mathsf{sign}(x, \mathsf{sk}_A, y)).\mathsf{out}(c^{\mathsf{check}}_{\mathsf{pk}_A,\mathsf{pk}_A}, x) \\ \mid\ !\mathsf{in}(c^{\mathsf{check}}_{\mathsf{pk}_B,\mathsf{pk}_B}, \mathsf{sign}(x, \mathsf{sk}_B, y)).\mathsf{out}(c^{\mathsf{check}}_{\mathsf{pk}_B,\mathsf{pk}_B}, x) \\ \mid\ !\mathsf{in}(c^{\mathsf{check}}_{\mathsf{pk}_C,\mathsf{pk}_C}, \mathsf{sign}(x, \mathsf{sk}_C, y)).\mathsf{out}(c^{\mathsf{check}}_{\mathsf{pk}_C,\mathsf{pk}_C}, x) \end{array}$$

4.2.3. Transformation $\mathcal{T}_{\mathsf{fwd}}$. Thanks to Lemmas 4.12 and 4.14, we are now able to formally define our transformation that gets rid of a fully active attacker. For every pair of protocols (P, Q) in $\mathcal{C}_{\mathsf{pp}}$, we consider

$$\mathcal{T}_{\mathsf{fwd}}(P, Q) = \{\mathcal{T}_{\mathsf{oracle}}(P', Q') \mid (P', Q') \in \mathcal{T}_{\mathsf{key}}(P, Q)\}$$

Combination of the two previous results yields to the desired result.

PROPOSITION 4.16. *Let P and Q be two protocols in $\mathcal{C}_{\mathsf{pp}}$. We have that:*

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

5. ENCODING PROTOCOLS INTO REAL-TIME GPDAS

We first introduce the notion of real-time generalized pushdown automaton (GPDA) (see Section 5.1) before explaining in details (see Sections 5.2 and 5.3) our encoding from protocols to real-time generalized pushdown automata. More precisely, for any process $P \in \mathcal{C}_{\mathsf{pp}}$, we show that it is possible to define a polynomial-sized real-time generalized pushdown automaton $\mathcal{A}_P$ such that trace equivalence w.r.t. the forwarder semantics coincides with language equivalence of the two corresponding automata.

THEOREM 5.1. *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 proof of this theorem consists of three main steps.

(1) First, we provide a new characterization of trace equivalence w.r.t. the forwarder semantics. Intuitively, we show that it is not necessary to consider all possible tests

We illustrate the different steps of our translation of protocols to automata using a (mock) ping-pong protocol P_{toy}. We define $\mathsf{io}(c, R, \mathsf{w}) \stackrel{\mathsf{def}}{=} \mathsf{in}(c, R).\mathsf{out}(c, \mathsf{w})$.

$$\begin{aligned} P_{\mathsf{toy}} = & \\ & \mid\ !\ \mathsf{in}(c_1, \mathsf{start}).\mathsf{new}\ r_1.\mathsf{out}(c_1, \mathsf{senc}(\mathsf{a}, \mathsf{k}_2, r_1)) \\ & \mid\ !\ \mathsf{in}(c_2, \mathsf{senc}(x, \mathsf{k}_2, z_1)).\mathsf{new}\ r_1.\mathsf{out}(c_2, \mathsf{senc}(x, \mathsf{k}_1, r_1)) \\ & \mid\ !\ \mathsf{in}(c_3, x).\mathsf{new}\ r_1.\mathsf{out}(c_3, \mathsf{senc}(x, \mathsf{k}_2, r_1)) \\ & \mid\ !\ \mathsf{in}(c_4, \mathsf{senc}(\mathsf{senc}(x, \mathsf{k}_1, z_1), \mathsf{k}_2, z_2)).\mathsf{new}\ r_1, r_2.\mathsf{out}(c_4, \mathsf{senc}(\mathsf{senc}(x, \mathsf{k}_2, r_1), \mathsf{k}_1, r_2)) \\ & \mid\ !\ \mathsf{in}(c_5, \mathsf{senc}(\mathsf{senc}(x, \mathsf{k}_2, z_1), \mathsf{k}_1, z_2)).\mathsf{out}(c_5, x) \end{aligned}$$

For illustrative purpose, we consider different execution traces of this protocol. For instance, we have that $(\mathsf{tr}_1, \sigma_1) \in \mathsf{trace}_{\mathsf{fwd}}(P_{\mathsf{toy}})$ where:

— $\mathsf{tr}_1 = \mathsf{io}(c_1, \mathsf{start}, \mathsf{w}_1).\mathsf{io}(c_2, \mathsf{w}_1, \mathsf{w}_2).\mathsf{io}(c_3, \mathsf{w}_2, \mathsf{w}_3).\mathsf{io}(c_4, \mathsf{w}_3, \mathsf{w}_4).\mathsf{io}(c_5, \mathsf{w}_4, \mathsf{w}_5)$, and
— $\sigma_1 = \{\mathsf{w}_1 \triangleright \mathsf{senc}(\mathsf{a}, \mathsf{k}_2, r_1),\ \mathsf{w}_2 \triangleright \mathsf{senc}(\mathsf{a}, \mathsf{k}_1, r_2),\ \mathsf{w}_3 \triangleright \mathsf{senc}(\mathsf{senc}(\mathsf{a}, \mathsf{k}_1, r_2), \mathsf{k}_2, r_3)$
$\mathsf{w}_4 \triangleright \mathsf{senc}(\mathsf{senc}(\mathsf{a}, \mathsf{k}_2, r_4), \mathsf{k}_1, r_5),\ \mathsf{w}_5 \triangleright \mathsf{a}\}$.

This execution may be continued as follows:

— $\mathsf{tr}_2 = \mathsf{io}(c_3, \mathsf{w}_1, \mathsf{w}_6).\mathsf{io}(c_2, \mathsf{w}_6, \mathsf{w}_7).\mathsf{io}(c_5, \mathsf{w}_7, \mathsf{w}_8)$, and
— $\sigma_2 = \{\mathsf{w}_6 \triangleright \mathsf{senc}(\mathsf{senc}(\mathsf{a}, \mathsf{k}_2, r_1), \mathsf{k}_2, r_6),\ \mathsf{w}_7 \triangleright \mathsf{senc}(\mathsf{senc}(\mathsf{a}, \mathsf{k}_2, r_1), \mathsf{k}_1, r_7),\ \mathsf{w}_8 \triangleright \mathsf{a}\}$.

Let $\sigma_{1/2} = \sigma_1 \cup \sigma_2$. We have that $(\mathsf{tr}_1.\mathsf{tr}_2, \sigma_{1/2})$ is a trace of P_{toy} w.r.t. the forwarder semantics. We have that the test $\mathsf{w}_5 = \mathsf{w}_8$ is valid in $\sigma_{1/2}$. Indeed $\mathsf{w}_5\sigma_{1/2}\!\downarrow = \mathsf{w}_8\sigma_{1/2}\!\downarrow = \mathsf{a}$.

We have also that $(\mathsf{tr}'_1, \sigma'_1) \in \mathsf{trace}_{\mathsf{fwd}}(P_{\mathsf{toy}})$ with:

— $\mathsf{tr}'_1 = \mathsf{io}(c_1, \mathsf{start}, \mathsf{w}_1).\mathsf{io}(c_2, \mathsf{w}_1, \mathsf{w}_2).\mathsf{io}(c_3, \mathsf{w}_2, \mathsf{w}_3).\mathsf{io}(c_4, \mathsf{w}_3, \mathsf{w}_4).\mathsf{io}(c_3, \mathsf{w}_4, \mathsf{w}_5)$, and
— $\sigma'_1 = \{\mathsf{w}_1 \triangleright \mathsf{senc}(\mathsf{a}, \mathsf{k}_2, r_1),\ \mathsf{w}_2 \triangleright \mathsf{senc}(\mathsf{a}, \mathsf{k}_1, r_2),\ \mathsf{w}_3 \triangleright \mathsf{senc}(\mathsf{senc}(\mathsf{a}, \mathsf{k}_1, r_2), \mathsf{k}_2, r_3),$
$\mathsf{w}_4 \triangleright \mathsf{senc}(\mathsf{senc}(\mathsf{a}, \mathsf{k}_2, r_4), \mathsf{k}_1, r_5),\ \mathsf{w}_5 \triangleright \mathsf{senc}(\mathsf{senc}(\mathsf{senc}(\mathsf{a}, \mathsf{k}_2, r_4), \mathsf{k}_1, r_5), \mathsf{k}_2, r_6)\}$.

This execution may be continued as follows:

— $\mathsf{tr}'_2 = \mathsf{io}(c_4, \mathsf{w}_5, \mathsf{w}_6).\mathsf{io}(c_5, \mathsf{w}_6, \mathsf{w}_7).\mathsf{io}(c_4, \mathsf{w}_5, \mathsf{w}_8).\mathsf{io}(c_5, \mathsf{w}_8.\mathsf{w}_9)$,
— $\sigma'_2 = \{\mathsf{w}_6 \triangleright \mathsf{senc}(\mathsf{senc}(\mathsf{senc}(\mathsf{a}, \mathsf{k}_2, r_4), \mathsf{k}_2, r_7), \mathsf{k}_1, r_8),\ \mathsf{w}_7 \triangleright \mathsf{senc}(\mathsf{a}, \mathsf{k}_2, r_4),$
$\mathsf{w}_8 \triangleright \mathsf{senc}(\mathsf{senc}(\mathsf{senc}(\mathsf{a}, \mathsf{k}_2, r_4), \mathsf{k}_2, r'_7), \mathsf{k}_1, r'_8),\ \mathsf{w}_9 \triangleright \mathsf{senc}(\mathsf{a}, \mathsf{k}_2, r_4)$.

Let $\sigma'_{1/2} = \sigma'_1 \cup \sigma'_2$. We have that $(\mathsf{tr}'_1.\mathsf{tr}'_2, \sigma'_{1/2})$ is a trace of P_{toy} w.r.t. the forwarder semantics. We have that the test $\mathsf{w}_7 = \mathsf{w}_9$ is valid in $\sigma'_{1/2}$.

Fig. 3. Running example.

(when checking static equivalence). Indeed, our Lemma 5.8 states that it is sufficient to check for constant tests (that is, tests of the form $x = c$ where c is a constant) and some specific class of tests that we call *guarded* and *pulled-up*.

(2) Then we associate to processes $P, Q \in \mathcal{C}_{\mathsf{pp}}$ real-time GPDAs that check whether they satisfy the same constant tests (Lemma 5.9).
(3) And we associate to processes $P, Q \in \mathcal{C}_{\mathsf{pp}}$ real-time GPDAs that check whether they satisfy the same guarded tests (Lemma 5.11).

All along this section, we illustrate the definitions with the protocol displayed in Figure 3. This example should be read step by step, when reading the examples of this section.

5.1. Generalized pushdown automata

Language equivalence of deterministic pushdown automata (DPA) is known to be decidable [Sénizergues 2001]. We actually encode equivalence of protocols into a fragment of DPA: real-time GPDA with final-state acceptance. GPDA differ from deterministic pushdown automata (DPA) as they can unstack several symbols at a time. Real-time automata are automata that do not include epsilon-transitions. Formally, the class of real-time GPDA is defined as follows.

Definition 5.2. A *real-time GPDA* is a 7-tuple $\mathcal{A} = (Q, \Pi, \Gamma, q_0, \omega, Q_f, \delta)$ where Q is a finite set of states, $q_0 \in Q$ is an initial state, $Q_f \subseteq Q$ is a set of accepting states, Π is a finite input-alphabet, Γ is a finite stack-alphabet, ω is the initial stack symbol, and $\delta : (Q \times \Pi \times \Gamma_0) \to Q \times \Gamma_0$ is a 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 $q, q' \in Q$, $u, u', \gamma \in \Gamma^*$, $m \in \Pi^*$, $a \in \Pi$; we note $(qu\gamma, am) \rightsquigarrow_{\mathcal{A}} (q'uu', m)$ if $(q', u') = \delta(q, a, \gamma)$. The relation $\rightsquigarrow^*_{\mathcal{A}}$ is the reflexive and transitive closure of $\rightsquigarrow_{\mathcal{A}}$. For every qu, $q'u'$ in $Q\Gamma^*$ and $m \in \Pi^*$, we note $qu \xrightarrow{m}_{\mathcal{A}} q'u'$ if, and only if, $(qu, m) \rightsquigarrow^*_{\mathcal{A}} (q'u', \epsilon)$. For sake of clarity, a transition from q to q' reading a, popping γ from the stack and pushing u' will be denoted by $q \xrightarrow{a;\gamma/u'} q'$.

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

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

Note that the language is defined starting with the word $\omega\mathsf{start}$ in the stack.

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 [Sénizergues 2001]. Whether deciding equivalence of real-time GPDA could be easier than deciding equivalence of DPA is an open question.

5.2. Characterization of trace equivalence

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.

We first propose a new characterization of trace equivalence allowing us to restrict our attention to executions of P and valid tests that have a special form.

Given an execution trace (tr, σ) and an element w of a frame σ, we can extract from tr the sequence of actions that conducted to the production of this element w.

Definition 5.3. Let P be a protocol in $\mathcal{C}_{\mathsf{pp}}$, tr be a trace of P w.r.t. the forwarder semantics, *i.e.* such that $(\mathsf{tr}, \sigma) \in \mathsf{trace}_{\mathsf{fwd}}(P)$ for some σ, and w be a variable that occurs in tr. The *sequence associated to* w *in* tr, denoted $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w})$, is the subsequence of tr of the following form:

$$\mathsf{seq}_{\mathsf{tr}}(\mathsf{w}) = \mathsf{io}(c_{i_0}, \mathsf{start}, \mathsf{w}_{j_0}).\mathsf{io}(c_{i_1}, \mathsf{w}_{j_0}, \mathsf{w}_{j_1}) \ldots \mathsf{io}(c_{i_p}, \mathsf{w}_{j_{p-1}}, \mathsf{w}).$$

Example 5.4. Consider the protocol defined in Figure 3. Then,

— $\mathsf{seq}_{\mathsf{tr}_1.\mathsf{tr}_2}(\mathsf{w}_5) = \mathsf{tr}_1$;
— $\mathsf{seq}_{\mathsf{tr}_1.\mathsf{tr}_2}(\mathsf{w}_8) = \mathsf{io}(c_1, \mathsf{start}, \mathsf{w}_1).\mathsf{tr}_2$;
— $\mathsf{seq}_{\mathsf{tr}'_1.\mathsf{tr}'_2}(\mathsf{w}_7) = \mathsf{tr}'_1.\mathsf{io}(c_4, \mathsf{w}_5.\mathsf{w}_6).\mathsf{io}(c_5, \mathsf{w}_6, \mathsf{w}_7)$;
— $\mathsf{seq}_{\mathsf{tr}'_1.\mathsf{tr}'_2}(\mathsf{w}_9) = \mathsf{tr}'_1.\mathsf{io}(c_4, \mathsf{w}_5, \mathsf{w}_8).\mathsf{io}(c_5, \mathsf{w}_8, \mathsf{w}_9)$.

We consider some particular class of tests, called *pulled-up* tests.

Definition 5.5. Let P be a protocol in $\mathcal{C}_{\mathsf{pp}}$, $(\mathsf{tr}, \sigma) \in \mathsf{trace}_{\mathsf{fwd}}(P)$, and $\mathsf{w}, \mathsf{w}' \in dom(\sigma)$ such that:

(1) the test $\mathsf{w} = \mathsf{w}'$ is σ-valid, *i.e.* $\mathsf{w}\sigma = \mathsf{w}'\sigma$; and
(2) the test $\mathsf{w} = \mathsf{w}'$ is σ-guarded, *i.e.* the head symbol of $\mathsf{w}\sigma$ (or equivalently $\mathsf{w}'\sigma$) is in $\{\mathsf{senc}, \mathsf{aenc}, \mathsf{sign}\}$.

Let $\mathsf{io}(c_{i_0}, \mathsf{start}, \mathsf{w}_{j_0}) \ldots \mathsf{io}(c_{i_p}, \mathsf{w}_{j_{p-1}}, \mathsf{w}_{j_p})$ be the maximal common prefix of $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w})$ and $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w}')$. The test $\mathsf{w} = \mathsf{w}'$ is said to be *pulled-up* in (tr, σ) if $p = 0$, or $p \geq 1$ and $\mathsf{w}\sigma$ does not occur as a subterm in $\mathsf{w}_{j_0}\sigma, \ldots, \mathsf{w}_{j_{p-1}}\sigma$.

Intuitively, to perform a test $\mathsf{w} = \mathsf{w}'$, the attacker (who acts as a forwarder) relies on the protocol rules to produce successive outputs, and ultimately the ones stored in w and w'. The attacker may produce w and w' independently (the common prefix of $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w})$ and $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w}')$ is empty), and in such a case the test is pulled-up by definition. This is not, of course, always possible. In particular, a test $\mathsf{w} = \mathsf{w}'$ satisfying conditions (1) and (2) of the previous definition is necessarily a "forked" test, *i.e.* a test for which the common prefix of $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w})$ and $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w}')$ is not reduced to the empty sequence, and thus $p \geq 1$. Indeed, $\mathsf{w}\sigma$ is a term of the form $\mathsf{f}(u, \mathsf{k}, r)$ with some random r. Since nonces are uniquely generated, the variables w_i that generates it, *i.e.* the smallest i such that r occurs in $\mathsf{w}_i\sigma$, occurs both in $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w})$ and $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w}')$. For this kind of "forked" test, we can restrict the attacker to consider tests that are pulled-up, i.e. we consider tests for which the size of the common prefix between $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w})$ and $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w}')$ is reduced to the minimum. This can be done by duplicating some execution steps since all the branches are under a replication.

Example 5.6. Continuing our running example, we have that $\mathsf{w}_5 = \mathsf{w}_8$ is a test that is $\sigma_{1/2}$-valid but it is not $\sigma_{1/2}$-guarded since $\mathsf{w}_5\sigma_{1/2} = \mathsf{w}_8\sigma_{1/2} = \mathsf{a}$.

The test $\mathsf{w}_7 = \mathsf{w}_9$ is a test that is $\sigma'_{1/2}$-valid and $\sigma'_{1/2}$-guarded. Indeed, we have that $\mathsf{w}_7\sigma'_{1/2} = \mathsf{w}_9\sigma'_{1/2} = \mathsf{senc}(\mathsf{a}, \mathsf{k}_2, r_4)$. The maximal common prefix of $\mathsf{seq}_{\mathsf{tr}'_1.\mathsf{tr}'_2}(\mathsf{w}_7)$ and $\mathsf{seq}_{\mathsf{tr}'_1.\mathsf{tr}'_2}(\mathsf{w}_9)$ is actually

$$\mathsf{tr}'_1 = \mathsf{io}(c_1, \mathsf{start}, \mathsf{w}_1).\mathsf{io}(c_2, \mathsf{w}_1, \mathsf{w}_2).\mathsf{io}(c_3, \mathsf{w}_2, \mathsf{w}_3).\mathsf{io}(c_4, \mathsf{w}_3, \mathsf{w}_4).\mathsf{io}(c_3, \mathsf{w}_4, \mathsf{w}_5).$$

Actually, $\mathsf{w}_7\sigma'_{1/2}$ occurs as a subterm in $\mathsf{w}_4\sigma'_{1/2}$, thus the test $\mathsf{w}_7 = \mathsf{w}_9$ is not pulled-up in $(\mathsf{tr}'_1.\mathsf{tr}'_2, \sigma'_{1/2})$.

We are now able to state our characterization lemma. Intuitively, we show that for tests that are valid and guarded, it is sufficient to consider pulled-up tests. We first illustrate through an example how a test that is valid and guarded can be converted into a pulled-up one.

Example 5.7. Continuing Example 5.4, we consider the test $\mathsf{w}_7 = \mathsf{w}_9$ which is not pulled-up in $(\mathsf{tr}'_1.\mathsf{tr}'_2, \sigma'_{1/2})$. Consider the execution

$$\mathsf{tr}' = \mathsf{tr}'_1.\mathsf{io}(c_4, \mathsf{w}_5, \mathsf{w}_6).\mathsf{io}(c_5, \mathsf{w}_6, \mathsf{w}_7).\mathsf{io}(c_3, \mathsf{w}_4, \mathsf{w}_8).\mathsf{io}(c_4, \mathsf{w}_8, \mathsf{w}_9).\mathsf{io}(c_5, \mathsf{w}_9, \mathsf{w}_{10}).$$

This execution is almost similar to $\mathsf{tr}'_1.\mathsf{tr}'_2$. The main difference is that the computation perfomed at the end of tr'_1 using channel c_3 with input w_4 is duplicated. Both $\mathsf{io}(c_3, \mathsf{w}_4, \mathsf{w}_5)$ and $\mathsf{io}(c_3, \mathsf{w}_4, \mathsf{w}_8)$ occur in tr'. The resulting frame is:

$$\begin{aligned}\sigma'_1 \ \cup \ \{&\mathsf{w}_6 \triangleright \mathsf{senc}(\mathsf{senc}(\mathsf{senc}(\mathsf{a}, \mathsf{k}_2, r_4), \mathsf{k}_2, r_7), \mathsf{k}_1, r_8),\ \mathsf{w}_7 \triangleright \mathsf{senc}(\mathsf{a}, \mathsf{k}_2, r_4),\\ &\mathsf{w}_8 \triangleright \triangleright \mathsf{senc}(\mathsf{senc}(\mathsf{senc}(\mathsf{a}, \mathsf{k}_2, r_4), \mathsf{k}_1, r_5), \mathsf{k}_2, r'_6),\\ &\mathsf{w}_9 \triangleright \mathsf{senc}(\mathsf{senc}(\mathsf{senc}(\mathsf{a}, \mathsf{k}_2, r_4), \mathsf{k}_2, r'_7), \mathsf{k}_1, r'_8),\ \mathsf{w}_{10} \triangleright \mathsf{senc}(\mathsf{a}, \mathsf{k}_2, r_4)\}.\end{aligned}$$

The terms stored in w_5 and w_8 differ by their random seeds:

$$\mathsf{senc}(\mathsf{senc}(\mathsf{senc}(\mathsf{a}, \mathsf{k}_2, r_4), \mathsf{k}_1, r_5), \mathsf{k}_2, r_6) \text{ and } \mathsf{senc}(\mathsf{senc}(\mathsf{senc}(\mathsf{a}, \mathsf{k}_2, r_4), \mathsf{k}_1, r_5), \mathsf{k}_2, r'_6)$$

This frame is almost the same as $\sigma'_{1/2}$ with an additional element (w_8). The term stored in w_8 is the same as the one stored in w_5 up to the choice of some random seeds (r_6 is replaced by the fresh random r'_6). Moreover, the presence of this additional element leads us to reindex the following elements of the frame, and to replace some occurrences of r_6 with r'_6. It is important to note that the introduced randoms r'_6 and r'_8 could potentially break equality tests. They however do not appear anymore in the last outputted term stored in w_{10} that is checked for equality.

This example shows that when considering the trace $(\mathsf{tr}'_1.\mathsf{tr}'_2, \sigma'_{1/2})$, we may have to consider the test $\mathsf{w}_7 = \mathsf{w}_9$ which is not pulled-up. However, this test is essentially the same than the pulled-up test $\mathsf{w}_7 = \mathsf{w}_{10}$ issued from the trace given above.

The transformation explained in the previous example can be generalized to any protocol.

LEMMA 5.8. *Let P and Q be two protocols in $\mathcal{C}_{\mathsf{pp}}$, then $P \approx_{\mathsf{fwd}} Q$ if, and only if, the following four conditions are satisfied:*

— CONST$_P$: *For all $(\mathsf{tr}, \sigma_P) \in \mathsf{trace}_{\mathsf{fwd}}(P)$, there exists a frame σ_Q such that $(\mathsf{tr}, \sigma_Q) \in \mathsf{trace}_{\mathsf{fwd}}(Q)$ and for every $\mathsf{w}, \mathsf{w}' \in dom(\sigma_P)$ and for every constant $\mathsf{c} \in \Sigma_0 \cup \{\mathsf{start}\}$, $\mathsf{w}\sigma_P = \mathsf{w}'\sigma_Q = \mathsf{c}$ if, and only if, there exists a constant $\mathsf{c}' \in \Sigma_0 \cup \{\mathsf{start}\}$ such that $\mathsf{w}\sigma_Q = \mathsf{w}'\sigma_Q = \mathsf{c}'$.*

— CONST$_Q$: *Similarly swapping the roles of P and Q.*

— GUARDED$_P$: *For all $(\mathsf{tr}, \sigma_P) \in \mathsf{trace}_{\mathsf{fwd}}(P)$, there exists a frame σ_Q such that $(\mathsf{tr}, \sigma_Q) \in \mathsf{trace}_{\mathsf{fwd}}(Q)$ and every test that is σ_P-valid, σ_P-guarded, and pulled-up in (tr, σ_P) is also σ_Q-valid, σ_Q-guarded, and pulled-up in (tr, σ_Q).*

— GUARDED$_Q$: *Similarly swapping the roles of P and Q.*

PROOF. *(sketch)*

($\Rightarrow$) For this direction, when considering CONST$_P$, the only difficulty is to show that the test $\mathsf{w}\sigma_Q = \mathsf{w}'\sigma_Q$ leads to a constant c'. Actually, such a test can not lead to a guarded test since otherwise a replay of the entire sequence (this replay is possible since we consider a class of protocol that allows this) will lead to a different guarded term in Q and not in P (due to the presence of fresh randoms in guarded terms).

When considering GUARDED$_P$, the difficulty is to show that the test $\mathsf{w} = \mathsf{w}'$ is necessarily pulled-up in (tr, σ_Q). Let $\mathsf{pref} = \mathsf{io}(c_{i_0}, \mathsf{start}, \mathsf{w}_{j_0}) \ldots \mathsf{io}(c_{i_p}, \mathsf{w}_{j_{p-1}}, \mathsf{w}_{j_p})$ be the maximal common prefix of $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w})$ and $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w}')$. Since $\mathsf{w} = \mathsf{w}'$ is pulled-up in (tr, σ_P), we know that the first occurrence of $\mathsf{w}\sigma_P$ in $\mathsf{pref}\sigma_P$ is at the very end of the sequence. We can easily show that $\mathsf{w} = \mathsf{w}'$ is σ_Q-valid and σ_Q-guarded, and thus $\mathsf{w}\sigma_Q$ occurs also as a subterm in $\mathsf{pref}\sigma_Q$. The only problem is if $\mathsf{w}\sigma_Q$ occurs in $\mathsf{pref}\sigma_Q$ but not at the very end of this sequence. The idea is that in such a case, we can modify the trace (tr, σ_Q) and the test $\mathsf{w} = \mathsf{w}'$ to build $(\mathsf{tr}^*, \sigma_Q^*)$ and a new test $\mathsf{w}_* = \mathsf{w}'_*$ which will be pulled-up in $(\mathsf{tr}^*, \sigma_Q^*)$. The idea is to split the two sequences $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w})$ and $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w}')$ earlier without compromising the fact that the test will be valid in the resulting frame. This corresponds to the construction illustrated in Example 5.7. This trace tr^* is actually a witness of non-equivalence. Actually, the test $\mathsf{w}_* = \mathsf{w}'_*$ is *a fortiori* not valid on the P side, and this contradicts our hypothesis $P \approx_{\mathsf{fwd}} Q$.

($\Leftarrow$) Actually, for this direction, assume that we have a witness of the fact that $P \not\approx Q$, *i.e.* a trace $(\mathsf{tr}, \sigma_P) \in \mathsf{trace}_{\mathsf{fwd}}(P)$, a trace $(\mathsf{tr}, \sigma_Q) \in \mathsf{trace}_{\mathsf{fwd}}(Q)$, and a test $\mathsf{w} = \mathsf{w}'$ that is σ_P-valid but not σ_Q-valid. In case the resulting term is a constant, we easily conclude that CONST$_P$ fails. Otherwise, it means that $\mathsf{w} = \mathsf{w}'$ is σ_P-guarded. In order to show that GUARDED$_P$ fails, we have to ensure that the test $\mathsf{w} = \mathsf{w}'$ is pulled-up w.r.t. (tr, σ_P). Since, this is not necessarily the case, we have to build another trace $(\mathsf{tr}^*, \sigma_P^*)$ that will

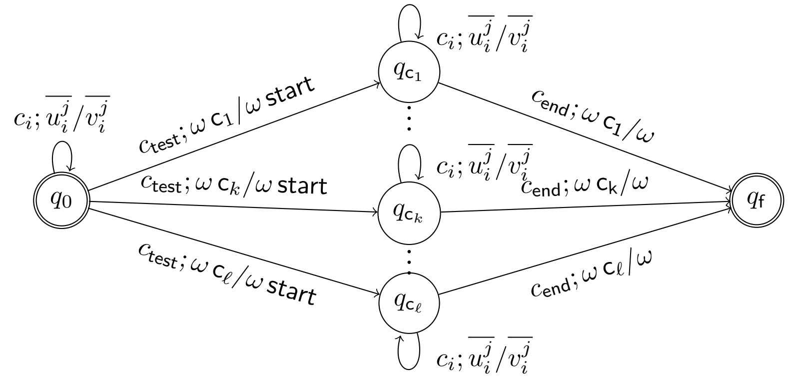


Fig. 4. Automaton $\mathcal{A}^P_{\text{CONST}}$

lead us to a pulled-up test. Roughly, the transformation consists in splitting the two sequences $\mathsf{seq_{tr}}(\mathsf{w})$ and $\mathsf{seq_{tr}}(\mathsf{w}')$ earlier without compromising the fact that the test will be valid in the resulting frame. Actually, such a transformation can not transform a test that was not valid in a valid one, thus this test is still not valid for Q and it is still a witness of non-equivalence, but a pulled-up one allowing us to conclude. □

The detailed proof can be found in Appendix C.1.

5.3. From trace equivalence to language equivalence

Our goal is to associate an automaton $\mathcal{A}_P$ to a protocol P such that $\mathcal{A}_P$ recognizes the words (a sequence of channels) that correspond to a possible execution of the protocol. The stack of the automaton $\mathcal{A}_P$ is used to store a (partial) representation of the last outputted term. This first requires to convert a term into a word.

Given an input term or an output term u (see Section 3.1), we define inductively $\bar{u}$ in the following way:

$$\begin{cases} \bar{u} = \bar{v}.\mathsf{k} & \text{if } u = \mathsf{f}(v, \mathsf{k}, r) \text{ and } \mathsf{f} \in \{\mathsf{senc}, \mathsf{aenc}, \mathsf{sign}\} \\ \bar{\mathsf{c}} = \omega\mathsf{c} & \text{for any constant } \mathsf{c} \in \Sigma_0 \cup \{\mathsf{start}\} \\ \bar{x} = \epsilon & \text{for any variable } x \end{cases}$$

where ϵ denotes the empty word. Note that, using this representation, random seeds are not part of the encoding. We denote by $||u||$ the *height* of the term u which is equal to the number of occurrence of senc, aenc, and sign in u.

We now consider an arbitrary ping-pong protocol P (using the same notation as the one introduced in Section 3):

$$P \stackrel{\mathsf{def}}{=} \Big|_{i=1}^{n} \Big|_{j=1}^{p_i} !\mathsf{in}(c_i, u_i^j).\mathsf{new}\ r_1.\ldots.\mathsf{new}\ r_{k_i^j}.\,\mathsf{out}(c_i, v_i^j) \qquad (*)$$

In the remaining of the section, we denote by Σ_0^P the finite set of constants of $\Sigma_0 \cup \{\mathsf{start}\}$ that actually occur in the protocol P.

5.3.1. Encoding of the conditions CONST_P *and* CONST_Q. We first build an automaton that recognizes tests of the form $\mathsf{w} = \mathsf{w}'$ such that the corresponding term is actually a constant. We define $\mathcal{A}^P_{\text{CONST}}$ as follows:

$$\mathcal{A}^P_{\text{CONST}} = (\{q_0, q_f\} \cup \{q_\mathsf{c} \mid \mathsf{c} \in \Sigma_0^P\}, \{c_1, \ldots, c_n\} \cup \{c_{\mathsf{test}}, c_{\mathsf{end}}\}, \Sigma_0^P, q_0, \omega, \{q_0, q_f\}, \delta)$$

where the transition function δ is defined as follows:

(1) for every $q \in \{q_0\} \cup \{q_\mathsf{c} \mid \mathsf{c} \in \Sigma_0^P\}$, for every $i \in \{1, \ldots, n\}$, for every $j \in \{1, \ldots, p_i\}$, there is a transition $q \xrightarrow{c_i;\, \bar{u}^i_j/\overline{v^i_j}} q$;

(2) for every constant c, there is a transition $q_0 \xrightarrow{c_\mathsf{test};\, \omega\, \mathsf{c}/\omega\, \mathsf{start}} q_\mathsf{c}$;

(3) for every constant c, there is a transition $q_\mathsf{c} \xrightarrow{c_\mathsf{end};\, \omega\, \mathsf{c}/\omega} q_\mathsf{f}$.

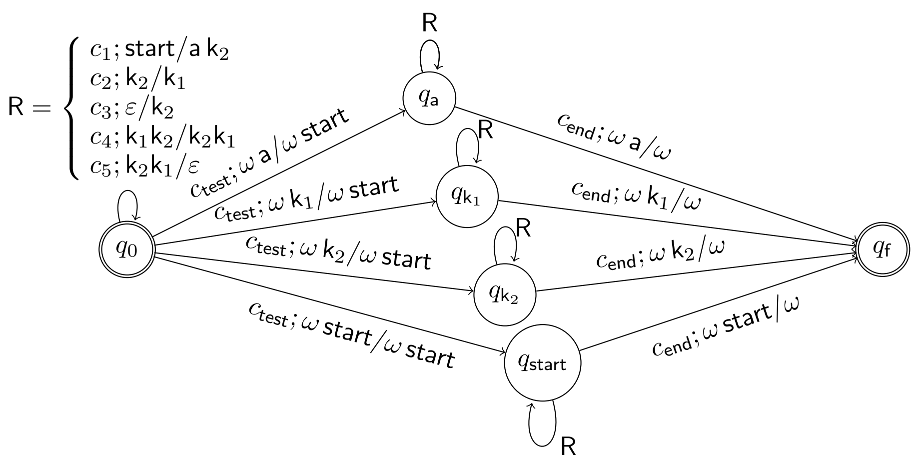
The automaton is depicted in Figure 4. Intuitively, the basic building blocks (e.g. q_0 with the transitions from q_0 to itself) mimic an execution of P where each input is fed with the last outputted term. Then, to recognize the tests of the form $\mathsf{w} = \mathsf{w}'$ that are true in such an execution, it is sufficient to memorize the constant c that is associated to w (adding a new state q_c), and to see whether it is possible to reach a state where the stack contains c again. More formally, we have the following result.

LEMMA 5.9. *Let P and Q be two protocols in $\mathcal{C}_\mathsf{pp}$, the two real-time GPDA $\mathcal{A}^P_{\text{CONST}}$ and A^Q_{CONST} are such that:*

P and Q satisfy conditions CONST_P and CONST_Q iff $\mathcal{L}(\mathcal{A}^P_{\text{CONST}}) = \mathcal{L}(\mathcal{A}^Q_{\text{CONST}})$.

The proof can be found in Appendix C.2.

Example 5.10. Going back to our running example, *i.e.* the protocol P described in Figure 3, the automaton $\mathcal{A}^P_{\text{CONST}}$ is depicted below:



The word that represents the trace $(\mathsf{tr}_1.\mathsf{tr}_2, \sigma_{1/2})$ and the test $\mathsf{w}_5 = \mathsf{w}_8$ as given in Figure 3 is: $c_1c_2c_3c_4c_5c_\mathsf{test}c_1c_3c_2c_5c_\mathsf{end}$. The fact that this test is a valid one that leads to a constant a means that the word will be accepted by the automaton given above. The corresponding run goes through the state q_a and halts in state q_f.

$\mathcal{A}^P_{\text{CONST}}$ has a number of states polynomial in the number of constants in P, and for each state a number of transitions linear in the number of branches in P. Thus, $\mathcal{A}^P_{\text{CONST}}$ is of size polynomial with respect to the size of P.

5.3.2. Encoding of the conditions GUARDED_P *and* GUARDED_Q. Capturing tests that lead to non-constant symbols (*i.e.* terms of the form $\mathsf{f}(u, \mathsf{k}, r)$ with $\mathsf{f} \in \{\mathsf{senc}, \mathsf{aenc}, \mathsf{sign}\}$) 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,

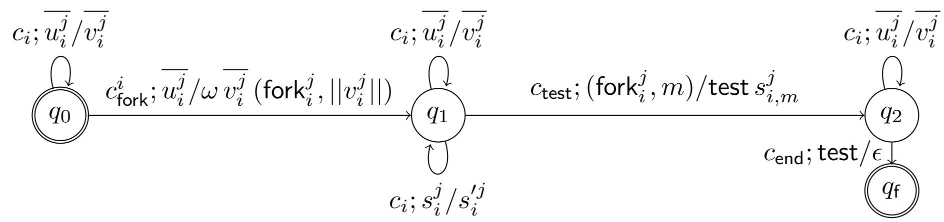


Fig. 5. Automaton $\mathcal{A}^P_{\text{GUARDED}}$

while they are forgotten in our encoding. We rely on our characterization introduced in Section 5.2 and we construct a more complex automaton that uses some special track symbols to encode when randomized ciphertexts may be reused.

More precisely, we consider:

— $\Pi = \{c_1, \dots, c_n, c_{\mathsf{test}}, c_{\mathsf{end}}\} \cup \{c^i_{\mathsf{fork}} \mid 1 \leq i \leq n\}$, and
— $\Gamma = \Sigma_0^P \cup \{\mathsf{test}\} \cup \{(\mathsf{fork}_i^j, k) \mid 1 \leq i \leq n,\ 1 \leq j \leq p_i,\ \text{and}\ 1 \leq k \leq ||u_i^j||\}$.

Note that n and p_i are induced by the definition of protocol P (see equation $(*)$). The input alphabet contains the channel names $c_1, \dots, c_n$, plus some additional symbols, denoted $c^1_{\mathsf{fork}}, \dots, c^n_{\mathsf{fork}}$, that will be used once and whose purpose will be to mark the end of the common prefix between $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w})$ and $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w}')$.

The stack-alphabet is more involved. We still have one symbol per constant in Σ_0^P, and a special symbol test that will be put on top of the stack when the stack contains the target term (*i.e.* $\mathsf{w}\sigma$). In such an automaton, the idea is to consider pulled-up tests only. The tile (fork_i^j, k) is placed on the stack when the automaton has finished to build the term corresponding to the left hand side of a pulled-up test.

The transition function δ is defined as follows:

(1) for $q \in \{q_0, q_1, q_2\}$, for every $i \in \{1, \dots, n\}$ and $j \in \{1, \dots, p_i\}$, there is a transition $q \xrightarrow{c_i;\, \overline{u_i^j}/\overline{v_i^j}} q$;
(2) for every $i \in \{1, \dots, n\}$ and $j \in \{1, \dots, p_i\}$ such that $||v_i^j|| \geq 1$, there is a transition
$$q_0 \xrightarrow{c^i_{\mathsf{fork}};\, \overline{u_i^j}/\omega\, \overline{v_i^j}\ (\mathsf{fork}_i^j, ||v_i^j||)} q_1$$
(3) for every $i \in \{1, \dots, n\}$ and $j \in \{1, \dots, p_i\}$, for every $i' \in \{1, \dots, n\}$ and $j' \in \{1, \dots, p_{i'}\}$, for every m such that $1 < m \leq ||v_{i'}^{j'}||$, and for every subterm u_0 of u_i^j of height $k \in \{1, \dots, m-1\}$ such that $\overline{u_i^j} = \overline{u_0}.s'$ there is a transition
$$q_1 \xrightarrow{c_i;\, \overline{u_0}.(\mathsf{fork}_{i'}^{j'}, m).s'/(\mathsf{fork}_{i'}^{j'}, m-k)\overline{v_i^j}} q_1.$$
(4) for every $i \in \{1, \dots, n\}$ and $j \in \{1, \dots, p_i\}$, for every m such that $1 \leq m \leq ||v_i^j||$, there is a transition $q_1 \xrightarrow{c_{\mathsf{test}};\, (\mathsf{fork}_i^j, m)/\mathsf{test}\ s^j_{i,m}} q_2$ where $s^j_{i,m}$ is the suffix of length $||v_i^j|| - m$ of $\overline{v_i^j}$.
(5) there is a transition $q_2 \xrightarrow{c_{\mathsf{end}};\, \mathsf{test}/\epsilon} q_{\mathsf{f}}$.

The loop in q_0 (item 1) 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 transitions $q_0 \xrightarrow{c^i_{\mathsf{fork}};\, z/z'} q_1$ (item 2) enable her to mark a fork when building a test in her frame with a particular stack symbol fork_i^j, enriched with some information. Intuitively, the part of the execution that is performed until here should correspond to the maximal prefix shared between the sequences $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w})$ and $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w}')$. 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 stack symbol (fork_i^j, k) which can only go down on the stack for at most $k-1$ times. When the symbol (fork_i^j, k) appears on top of the stack, the attacker may decide that she has built the first part of a pulled-up test. Then test will be put on the top of the stack and a part of the stack (following the instructions memorized in the symbol (fork_i^j, k)) will be regenerated. The idea is that the stack has to contain the same term as the one stored just after forking. Then the attacker tries to build the second member of the test. If this second term manages to end up exactly as the previous one (the position in the stack is marked using the tile test), an equality is reached and the word is recognized by the automata, witnessing the equality induced by the pulled-up test.

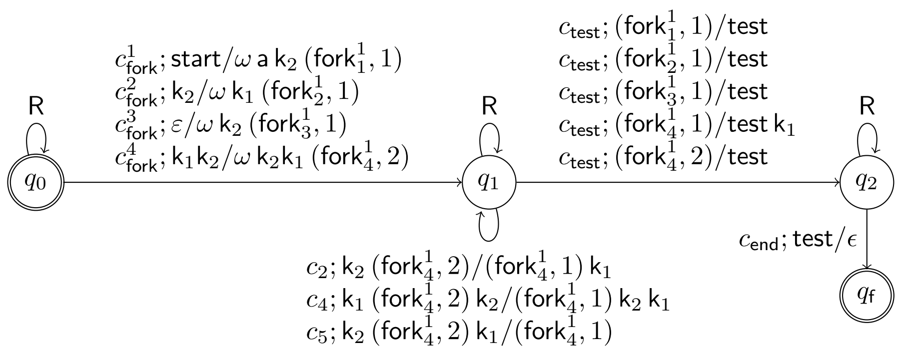
What remains now is to prove that P and Q satisfy conditions GUARDED_P and GUARDED_Q if, and only if, $\mathcal{L}(\mathcal{A}^P_{\text{GUARDED}}) = \mathcal{L}(\mathcal{A}^Q_{\text{GUARDED}})$. This is formally stated in the following lemma.

LEMMA 5.11. *Let P and Q be two protocols in $\mathcal{C}_{\mathsf{pp}}$, the two real-time GPDA $\mathcal{A}^P_{\text{GUARDED}}$ and A^Q_{GUARDED} are such that:*

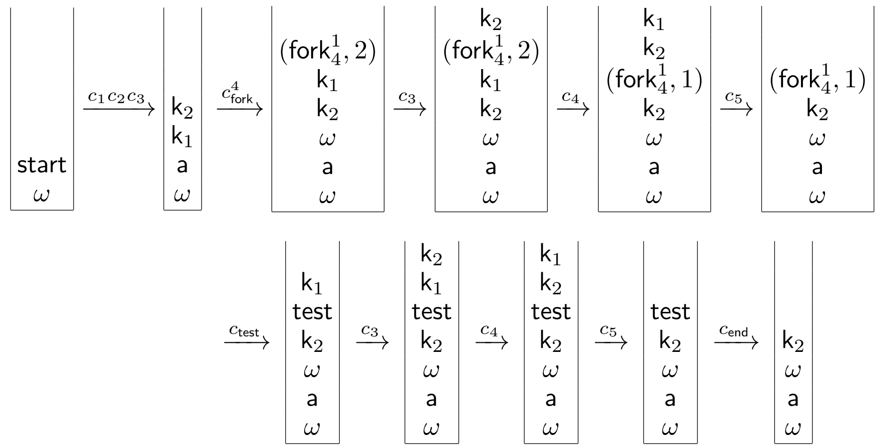
$$P \text{ and } Q \text{ satisfy conditions } \text{GUARDED}_P \text{ and } \text{GUARDED}_Q \text{ iff}$$
$$\mathcal{L}(\mathcal{A}^P_{\text{GUARDED}}) = \mathcal{L}(\mathcal{A}^Q_{\text{GUARDED}}).$$

The proof can be found in Appendix C.2.

Example 5.12. Going back to our running example, *i.e.* the protocol P described in Figure 3, the automaton $\mathcal{A}^P_{\text{GUARDED}}$ is depicted below.



The set of transitions R is the one defined in Example 5.10. The situation where the stack symbol (fork_i^j, k) goes down occurs for instance when considering the word $c_1\ c_2\ c_3\ c^4_{\mathsf{fork}}\ c_3\ c_4\ c_5\ c_{\mathsf{test}}\ c_3\ c_4\ c_5$. The evolution of the stack during the run of the automaton is depicted below. On the second line, we can see that this symbol goes down and k goes from 2 to 1.



The trace $(\mathsf{tr}, \sigma) \in \mathsf{trace}_{\mathsf{fwd}}(P)$ and the pulled-up test $\mathsf{w} = \mathsf{w}'$ that correspond to this execution is the ones introduced in Example 5.7, *i.e.* tr' together with the test $\mathsf{w}_7 = \mathsf{w}_{10}$.

We can notice that up to the special stack-symbols, namely test and (fork_i^j, k), the contents of the stack after reading c^i_{fork} (here $i = 4$) and c_{test} are the same. The stack actually represents the term obtained after executing the common prefix shared between $\mathsf{seq}_{\mathsf{tr}'}(\mathsf{w}_7)$ and $\mathsf{seq}_{\mathsf{tr}'}(\mathsf{w}_{10})$, *i.e.* $\mathsf{senc}(\mathsf{senc}(\mathsf{a}, \mathsf{k}_2, r_4), \mathsf{k}_1, r_5)$ stored in w_4. We have also that the contents of the stack before reading c_{test} and after reading c_{end} are also the same (up to some special stack symbols). They actually represent the terms stored respectively in w_7 and w_{10}.

Note that $\mathcal{A}^P_{\text{GUARDED}}$ 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, $\mathcal{A}^P_{\text{GUARDED}}$ is of size polynomial with respect to the size of P.

6. FROM LANGUAGE EQUIVALENCE TO TRACE EQUIVALENCE

We have seen how to encode trace equivalence between processes in $\mathcal{C}_{\mathsf{pp}}$ into langage equivalence between real-time GPDA. The two problems are actually *equivalent*. Indeed, in this section, we show that we can conversely encode any real-time GPDA $\mathcal{A}$ into a process $P_{\mathcal{A}}$ in $\mathcal{C}_{\mathsf{pp}}$ such that $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ implies $P_{\mathcal{A}} \sqsubseteq P_{\mathcal{B}}$.

Consider an automaton $\mathcal{A} = (Q, \Pi, \Gamma, q_0, \omega, Q_f, \delta)$. The process $P_{\mathcal{A}}$ associated to $\mathcal{A}$ is built using symmetric encryption only. For the purpose of the encoding, we consider the following constants of sort SymKey:

— for each $q \in Q$, we denote q its counterpart in Σ_0;
— for each $\alpha \in \Gamma$, we denote k_α its counterpart in Σ_0;
— a constant $\mathsf{k}_{\mathsf{well}}$.

Let also c_0, c_a, c_f with $a \in \Pi$ be constant symbols of sort $\mathsf{channel}$ in $\mathcal{C}h$. Words in Γ^*, *i.e.* stacks, will be represented through nested encryptions with 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 $\overline{x.u}$:

— either the term $\mathsf{senc}(\dots\mathsf{senc}(x, \mathsf{k}_{\alpha_1}, z_1)\dots, \mathsf{k}_{\alpha_p}, z_p)$ where z_1 through z_p are variables used for nonces when $\overline{x.u}$ is used in as an input pattern;
— or the term $\mathsf{senc}(\dots\mathsf{senc}(x, \mathsf{k}_{\alpha_1}, r_1)\dots, \mathsf{k}_{\alpha_p}, r_p)$ where r_1 through r_p are fresh randoms when $\overline{x.u}$ is used as an output pattern.

Below, we use $\mathsf{new}\,\tilde{r}$ as shortcut for $\mathsf{new}\,r_1.\ldots.\mathsf{new}\,r_p$ such that the sequence will bind every nonce occurring in the following output.

The stack of the automaton $\mathcal{A}$ is encoded as a pile of encryptions (where each key encodes a letter of the stack). Then, upon receiving such a pile of encryptions encrypted by q on channel c_a, the process $P_\mathcal{A}$ will mimic the transition of $\mathcal{A}$ that is triggered when the automaton is at state q upon reading a with the stack corresponding to that pile of encryptions.

More formally, the process $P_\mathcal{A}$ is defined as follows:

$$
\begin{array}{rll}
P_\mathcal{A} \stackrel{\mathsf{def}}{=} & !\,\mathsf{in}(c_0, \mathsf{start}).\mathsf{new}\ \tilde{r}.\mathsf{out}(c_0, \mathsf{senc}(\mathsf{senc}(\mathsf{start}, \mathsf{k}_\omega, r_1), \mathsf{k}_{\mathsf{start}}, r_2), q_0, r_3)) & (0)\\
& \mid\ !\,\mathsf{in}(c_a, \mathsf{senc}(\overline{x.u}, q, z)).\mathsf{new}\ \tilde{r}.\mathsf{out}(c_a, \mathsf{senc}(\overline{x.v}, q', r)) & (1)\\
& \mid\ !\,\mathsf{in}(c_a, \mathsf{senc}(\overline{x.u'}, q, z)).\mathsf{new}\ r.\mathsf{out}(c_a, \mathsf{senc}(\mathsf{start}, \mathsf{k}_{\mathsf{well}}, r)) & (1a)\\
& \mid\ !\,\mathsf{in}(c_a, \mathsf{senc}(\mathsf{start}, \mathsf{k}_{\mathsf{well}}, z)).\mathsf{new}\ r.\mathsf{out}(c_a, \mathsf{senc}(\mathsf{start}, \mathsf{k}_{\mathsf{well}}, r)) & (1b)\\
& \mid\ !\,\mathsf{in}(c_f, \mathsf{senc}(x, q_f, z)).\mathsf{new}\ r.\mathsf{out}(c_f, \mathsf{senc}(\mathsf{start}, q_f, r)) & (2)
\end{array}
$$

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 $(q', v) = \delta(q, a, u)$. Lastly, u' ranges over $U'_{q,a} \stackrel{\mathsf{def}}{=} \alpha \cdot SS_{q,a} \smallsetminus S_{q,a}$ where $S_{q,a}$ (resp. $SS_{q,a}$) is the set that contains suffixes (resp. strict suffixes) of some u with $(q, a, u) \in dom(\delta)$. This set $U'_{q,a}$ corresponds intuitively to the set of shortest words which are not suffixes of any word in $\{u \mid (q, a, u) \in dom(\delta)\}$, and, thus the shortest words to unstack to be sure that no transition from q reading a is possible in the automaton.

Example 6.1. Consider a real-time GPDA such that $\Gamma = \{\alpha, \beta, \gamma, \omega\}$, $q \in Q$, and $a \in \Pi$. Assume that $\{u \mid (q, a, u) \in dom(\delta)\} = \{\beta\alpha, \beta\alpha\alpha\}$. We have $SS_{q,a} = \{\epsilon, \alpha, \alpha\alpha\}$, and $S_{q,a} = SS_{q,a} \cup \{\beta\alpha, \beta\alpha\alpha\}$. Thus, we have that:

$$U'_{q,a} = \{\omega, \beta, \gamma, \omega\alpha, \gamma\alpha, \omega\alpha\alpha, \alpha\alpha\alpha, \gamma\alpha\alpha\}.$$

In the encoding above, the branches (0) and (1) mimic the behaviour of the automaton $\mathcal{A}$. Branch (2) is triggered in case a final state q_f is reached. In case we are considering a behaviour that is not authorized by the automaton, we obtain a message encrypted with $\mathsf{k}_{\mathsf{well}}$ through branches $(1a)$. Then branches $(1b)$ allow to pursue the execution of the protocol outputting messages that look fresh.

LEMMA 6.2. *The protocol $P_\mathcal{A}$ described above is in $\mathcal{C}_{\mathsf{pp}}$ and of size polynomial w.r.t. $\mathcal{A}$.*

PROOF. First, note that because $dom(\delta)$ is finite, as the automaton is finitely described, the sets $\{u \mid (q, a, u) \in dom(\delta)\}$ and $U'_{q,a}$ 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 $s \in \Gamma^*$:

— either there exists a unique suffix u of s such that $(q, a, u) \in dom(\delta)$;
— or there exists a unique suffix u' of s such that $u' \in U'_{q,a}$,

and this disjunction is exclusive. This allows us to ensure that condition (2) of Definition 3.1 is satisfied, and thus $P_\mathcal{A}$ belongs to $\mathcal{C}_{\mathsf{pp}}$.

Regarding the size of the protocol, the only non-trivial point is to check that the number of branches $(1a)$ is polynomially bounded. Let $q \in Q$, and $a \in \Pi$, and assume that the maximal length of a word u in a transition $q \xrightarrow{a;u/v} q'$ of the automaton is $\ell_{q,a}$, we have that the number of branches $(1a)$ for state q and letter a is bounded bounded by $\ell_{q,a} \times \#\Gamma \times \#\{u \mid (q,a,u) \in dom(\delta)\}$ where $\#S$ is the cardinality of set S. This allows us to conclude. $\square$

This polynomial encoding preserves inclusion.

PROPOSITION 6.3. *Let $\mathcal{A}$ and $\mathcal{B}$ be two real-time GPDA. We have that:*

$$\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B}) \iff P_{\mathcal{A}} \sqsubseteq P_{\mathcal{B}}.$$

PROOF. Let $\mathcal{A} = (Q, \Pi, \Gamma, q_0, \omega, Q_f, \delta)$ and $\mathcal{B} = (Q', \Pi, \Gamma', q'_0, \omega, Q'_f, \delta')$. We show the two implications separately.
($\Leftarrow$) Assume that there exists a word $t \in \mathcal{L}(\mathcal{A}) \smallsetminus \mathcal{L}(\mathcal{B})$. We will build a trace $(\mathsf{tr}, \phi) \in \mathsf{trace}(P_{\mathcal{A}})$ such that there exists no trace $(\mathsf{tr}, \psi) \in \mathsf{trace}(P_{\mathcal{B}})$ allowing us to conclude that $P_{\mathcal{A}} \not\sqsubseteq P_{\mathcal{B}}$. To build (tr, ϕ), we will mimick the behaviour of $\mathcal{A}$ when reading t. The first branch to use is (0), enabling the attacker to activate other branches of the process $P_{\mathcal{A}}$. As $t \in \mathcal{L}(\mathcal{A})$ and $\mathcal{A}$ is deterministic, there exists a unique sequence of transitions leading to an accepting state $q_f \in Q_f$. For every such transition the attacker will activate the corresponding branch (1) in $P_{\mathcal{A}}$. If $t = a_1 \ldots a_n$, we define (tr, ϕ) as follows:

$$\mathsf{tr} = \ \mathsf{io}(c_0, \mathsf{start}, \mathsf{w}_1).\mathsf{io}(c_{a_1}, \mathsf{w}_1, \mathsf{w}_2) \ldots \mathsf{io}(c_{a_n}, \mathsf{w}_n, \mathsf{w}_{n+1}).\mathsf{io}(c_f, \mathsf{w}_{n+1}, \mathsf{w}_{n+2})$$

and ϕ is defined as expected given our semantics. Because of the definition of the branch (1), 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 (2) and output the message $\mathsf{senc}(\mathsf{start}, q_f, r)$. As $t \notin \mathcal{L}(\mathcal{B})$, the corresponding sequence of transitions in $\mathcal{B}$ does not lead to any accepting state:

— either at some point of the execution of the automaton a transition from state q reading a is not possible with the current stack s. This means that there does not exist a suffix u of s such that $(q, a, u) \in dom(\delta')$, and thus, by definition of $U'_{q,a}$, there exists a suffix u' of s such that $u' \in U'_{q,a}$, enabling a transition $(1a)$ on channel c_a for the attacker, and every subsequent transition is done using branches $(1b)$,
— or the state reached in $\mathcal{B}$ after reading t is not an accepting state, *i.e.* not in Q'_f: 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 $(\mathsf{tr}, \psi) \in \mathsf{trace}(P_{\mathcal{B}})$ (for any ψ), thus $P_{\mathcal{A}} \not\sqsubseteq P_{\mathcal{B}}$.

($\Rightarrow$) First note that, for every frame ϕ (resp ψ) such that $(\mathsf{tr}, \phi) \in \mathsf{trace}(P_{\mathcal{A}})$ (resp. $(\mathsf{tr}, \psi) \in \mathsf{trace}(P_{\mathcal{B}})$), we have that ϕ (resp. ψ) is of the form

$$\{\mathsf{w}_1 \triangleright \mathsf{senc}(m_1, k_1, r_1), \ldots, \mathsf{w}_n \triangleright \mathsf{senc}(m_n, k_n, r_n)\}$$

where the k_i are non deducible and the r_i are "fresh" in the sense that they are all distinct and non deducible. This means that no equality (but the trivial ones) holds in such a frame. Now consider the shortest trace $(\mathsf{tr}, \phi) \in \mathsf{trace}(P_{\mathcal{A}})$, in terms of number of transitions, such that there exists no equivalent frame $(\mathsf{tr}, \psi) \in \mathsf{trace}(P_{\mathcal{B}})$. Since keys are non-deducible, we may assume w.l.o.g that $(\mathsf{tr}, \phi) \in \mathsf{trace}_{\mathsf{fwd}}(P_{\mathcal{A}})$. Because of the branches (1), $(1a)$ and $(1b)$ and in particular of the definition of $U'_{q,a}$, for any $q \in Q$, for any $a \in \Pi$, a transition of channel c_a is always possible, and we have seen that the resulting frames are necessarily in static equivalence. Thus, the only shortest trace where $P_{\mathcal{B}}$ will not be able to follow is when tr ends with an input/output on channel c_f. Let $\mathsf{w} \in dom(\phi)$ be the corresponding variable in the frame ϕ. Consider the subsequence $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w})$ of tr and more precisely the sequence of channels that occurs in this subsequence. Such a sequence is of the form: $c_0.c_{a_1} \ldots c_{a_n}.c_f$.

Let $v = a_1 \ldots a_n$. We have that v is a word of Π^*, and, in particular,

— $v \in \mathcal{L}(\mathcal{A})$: indeed, branches (1) in $P_{\mathcal{A}}$ faithfully represent transitions $(q, a, u) \in dom(\delta)$ and a branch (2) can only be fired if $q_f \in Q_f$.
— $v \notin \mathcal{L}(\mathcal{B})$: indeed branch (2) could not be fired, either $\mathcal{B}$ cannot read v or, after reading v, $\mathcal{B}$ is not in any state of Q'_f.

Hence $v \in \mathcal{L}(\mathcal{A}) \smallsetminus \mathcal{L}(\mathcal{B})$, proving that $\mathcal{L}(\mathcal{A}) \nsubseteq \mathcal{L}(\mathcal{B})$. □

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

7. IMPLEMENTATION

In this section, we detail our tool Cpp2dpa to convert protocols in $\mathcal{C}_{\mathsf{pp}}$ into GDPA, available online at

http://www.lsv.ens-cachan.fr/~chretien/cpp2dpa.php.

This tool takes two protocols in $\mathcal{C}_{\mathsf{pp}}$ as input, turn them into GDPA and, through the tool lAlBlC [Henry and Sénizergues 2013], outputs whether the two protocols were in equivalence, yielding a witness of non-equivalence in the negative case in the form of a sequence of channels leading to an attack. The tool focuses on the encoding as described in Section 5. In particular, we assume the prior steps of Section 4 were successfully applied to the pair of protocols; namely the bijection α as in Lemma 4.12 was successfully guessed and the oracles of Section 4.2.2 correctly added.

The tool Cpp2dpa is written in Python 3. From pairs of protocols in $\mathcal{C}_{\mathsf{pp}}$, it generates three pairs of *normalized determinisitic pushdown automaton*, instead of directly two pairs of GPDA (as described in Section 5). This was necessary so as to interface with lAlBlC, and involves no loss of generality, as the former are more expressive than our GDPA. The normalization process still has the inconvenient, in order to preserve the determinacy of the result, to output automata that may duplicate actions. More specifically, when necessary, the channels appearing in the potential witness of non-equivalence may be doubled. This technical detail does not impair the ability for the combined tool to prove equivalence or find witnesses, nevertheless.

7.1. Encoding pairs

Most protocols use pairs. While our formalism does not directly support pairs, we may encode a restricted kind of pairing, when there are only constants (such as identities) on the right. Formally, this amounts into encoding a pair $\langle t, a\rangle$, where t is a term and a some constant, by an encryption $\mathsf{senc}(t, a, r)$ for some random seed r. Provided constants used in concatenation are disjoint from constants used as keys, this encoding does not introduce any confusion. Note that since encryption is randomized, this pairing operator also differs as it is randomized.

7.2. Biometric passport

We are interested here in proving the unlinkability of the electronic passport protocol. A detailed specification of it can be found in [Arapinis et al. 2010]. Here, we only consider the passport's role and forget about the reader. The first case we consider is the flawed version corresponding to the French implementation of the passport, in which an attack arises from the ability for the attacker to observe whether a MAC check succeeds or not. As our framework does not directly enables us to deal with pairs of messages with their MAC, we model it by a signature: the attacker is able to obtain the plaintext of it (which amounts to retrieving the first component of the real pair) but cannot forge it (the attacker is not a priori able to forge a valid MAC). The resulting process is defined as follows.

$$\begin{array}{rll} P_A \stackrel{\mathsf{def}}{=} & !\,\mathsf{in}(c_1, \mathsf{start}).\mathsf{new}\,\tilde{r}'. & \\ & \quad \mathsf{out}(c_1, \mathsf{sign}(\mathsf{senc}(\mathsf{senc}(\mathsf{senc}(n_r, \mathsf{k}_r, r'_1), n_p, r'_2), \mathsf{k}_E, r'_3), \mathsf{mac}_{\mathsf{k}_m}, r'_4)) & (1) \\ & \mid\, !\,\mathsf{in}(c_2, \mathsf{sign}(\mathsf{senc}(x, \mathsf{k}_E, z_1), \mathsf{mac}_{\mathsf{k}_m}, z_2)).\mathsf{new}\,r_5.\mathsf{out}(c_2, \mathsf{sign}(x, \mathsf{mac}_{\mathsf{ok}}, r_5)) & (2a) \\ & \mid\, !\,\mathsf{in}(c'_2, \mathsf{sign}(\mathsf{senc}(x, n_p, z_1), \mathsf{mac}_{\mathsf{ok}}, z_2)).\mathsf{new}\,\tilde{r}''. & \\ & \quad \mathsf{out}(c'_2, \mathsf{sign}(\mathsf{senc}(\mathsf{senc}(x, n_p, r''_1), \mathsf{k}_p, r''_2), \mathsf{mac}_{\mathsf{k}_m}, r''_3)) & (2b) \end{array}$$

where $\mathsf{new}\,\tilde{r}$ is a shortcut of $\mathsf{new}\,r_1.\mathsf{new}\,r_2.\mathsf{new}\,r_3.\mathsf{new}\,r_4$ (and similarly for $\mathsf{new}\,\tilde{r}'$ and $\mathsf{new}\,\tilde{r}''$). The protocol is modeled through three rules. Branch (1) corresponds to a message from the current session, emitted by the reader. While the original protocol can check the authenticity of the MAC and the value of the nonce sent to the passport, our formalism requires us to separate this into two steps: branches $(2a)$ and $(2b)$. Branch $(2a)$ checks the validity of the MAC: if it is, it send a message signed with $\mathsf{mac}_{\mathsf{ok}}$. On the other hand, branch $(2b)$ checks the value of the nonce (*i.e.* n_p) and finally emits the last message of the protocol. To retrieve the attack, we introduce the message sent by the reader from a previous session with a new branch denoted (0):

$$\begin{array}{ll} !\,\mathsf{in}(c_0, \mathsf{start}).\mathsf{new}\,\tilde{r}. & \\ \quad \mathsf{out}(c_0, \mathsf{sign}(\mathsf{senc}(\mathsf{senc}(\mathsf{senc}(n_r^0, \mathsf{k}_r^0, r_1), n_p^0, r_2), \mathsf{k}_E, r_3), \mathsf{mac}_{\mathsf{k}_m}, r_4)) & (0) \end{array}$$

Another protocol P_B is obtained by replacing $\mathsf{mac}_{\mathsf{k}_m}$ by $\mathsf{mac}_{\mathsf{k}'_m}$ in branches (1), $(2a)$ and $(2b)$. Our tool Cpp2dpa can automatically check that $P_A \not\approx P_B$.

Another version P'_A is obtained by replacing branches $(2a)$ and $(2b)$ by the branch

$$\begin{array}{ll} !\,\mathsf{in}(c_2, \mathsf{sign}(\mathsf{senc}(\mathsf{senc}(x, n_p, z_1), \mathsf{k}_E, z_2), \mathsf{mac}_{\mathsf{k}_m}, z_3)).\mathsf{new}\,\tilde{r}''. & \\ \quad \mathsf{out}(c_2, \mathsf{sign}(\mathsf{senc}(\mathsf{senc}(x, n_p, r''_1), \mathsf{k}_E, r''_2), \mathsf{mac}_{\mathsf{k}_m}, r''_3)) & (2) \end{array}$$

The protocol P'_B is similarly defined, with $\mathsf{mac}_{\mathsf{k}'_m}$ instead of $\mathsf{mac}_{\mathsf{k}_m}$ in this branch (2). This version models the safe implementation of the protocol, where the success or failure of the MAC check is invisible to the attacker. Our tool Cpp2dpa can automatically prove that $P'_A \approx P'_B$.

7.3. Experiments

We have tested our tool Cpp2dpa on the running example as defined in Example 2.7 and Example 4.5; as well as on an encoding of the electronic passport protocol, described in Section 7.2 in two versions, unsafe and safe (see [Arapinis et al. 2010] for more details).

	Automata (in ms)	Grammars (in s)	Equivalence (in s)
Example 2.7	7.1	9.2	3462 (attack)
Example 4.5	7.0	3.1	9788 (proof)
Unsafe passport	7.1	23.2	4.89 (attack)
Safe passport	8.1	15.0	76.1 (proof)

The experiments were conducted on a Intel(R) Xeon(R) CPU X5650 @ 2.67GHz with 47 Go of RAM, using one core only. The first column corresponds to the cumuled time required to produce the different automata; the second one the time needed to convert the automata into grammars to be processed by 1A1B1C and the third one to the status of the equivalence (proof or witness of non-equivalence) and the cumuled time spent to prove the equivalence (when it is the case) or the execution time to find a witness of non-equivalence, when possible. There is non-equivalence as soon as one of our three pairs of automata are not in equivalence. Since we execute 1A1B1C in parallel for each of these three pairs, the execution time corresponds to the first pair that is found to

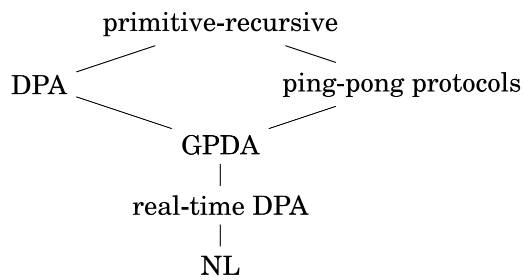


Fig. 6. Complexity bounds for equivalence of ping-pong protocols.

be not in equivalence. Converting the automata to grammars required an optimization of the built-in functionnality in `lAlBlC` in order to reach reasonable execution times. Other protocols were considered, namely variants of the Wide Mouthed Frog, Denning-Sacco and Private Authentication protocols. Unfortunately, for these ones, albeit the generation of automata was quick, it was impossible to prove (non-)equivalence in reasonable time with `lAlBlC`.

8. DISCUSSION AND CONCLUSION

We have shown a first decidability result for equivalence of (deterministic) ping-pong protocols for an unbounded number of sessions by reducing it to the equality of languages of deterministic, generalized, real-time pushdown automata (GPDA). We further show that deciding equivalence of ping-pong protocols is actually at least as hard as deciding equality of languages of GPDA. Complexity-wise, the situation is slightly less clear. While the reduction from GPDA to ping-pong protocols is polynomial, the reduction from ping-pong protocols to GPDA requires an exponential blow-up. Indeed, to get rid of the attacker, we guess a correspondance between the keys of P and Q, and exponentially many such correspondences should be checked. In addition, the complexity of equivalence of various classes of pushdown automata are not very well-known. 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 [Stirling 2002]. The lower bound comes from the fact that real-time deterministic pushdown automata are at least NL-hard [Boehm and Goeller 2011]. Whether equivalence of DPA (or even real-time GPDA) is e.g. at least NP-hard is unknown. The complexity hierarchy known so far for equivalence of ping-pong protocols is displayed in Figure 6.

Note that the complexity of GPDA and ping-pong protocols is actually quite close since the reduction from ping-pong protocols to GPDA is “just” exponential. Moreover, assume now that we consider only procedures that return a witness of non equivalence (if any). Then the complexity classes of GPDA and ping-pong protocols should actually coincide. Indeed, assume that there is a procedure for checking equivalence of GPDA that ends in time $f(n)$ where n is the size of the inputs, and that returns a witness when two automata are not in equivalence. This witness must be of size at most $f(n)$. Then given two ping-pong protocols P and Q, we would construct $\bar{P}$ and $\bar{Q}$ as defined in Lemma 4.12 step by step.

Instead of guessing the sets K and K', we would start from the emptysets $K = K' = \emptyset$. If $\bar{P} \not\approx \bar{Q}$, that is if $\mathcal{A}_{\bar{P}} \not\approx \mathcal{A}_{\bar{Q}}$, we consider a witness of non equivalence. Either it is a witness of $P \not\approx Q$ (and we are done), or there must exist a key k that is deducible in P and a corresponding key k' deducible with the same actions in Q. We start over with $K = \{\mathsf{k}\}$ and $K' = \{\mathsf{k}'\}$.

This algorithm has at most n steps and each step involve a call to the GPDA procedure ($\mathcal{A}_{\bar{P}} \approx \mathcal{A}_{\bar{Q}}$) and involves replaying a witness of size $f(n)$. This yields a procedure of complexity $O(f(n))$.

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 pairing is not straightforward. A direction is to use pushdown automata for which stacks are terms.

While we consider an unbounded number of sessions, we consider a fixed number of agents in our examples. We could model an unbounded number of agents, however, since our class considers protocols rules with at most one variable, we could consider at most one agent per rule with no key nor nonces, which would be very restrictive. Another direction is to study whether we can soundly bound the number of agents.

Our tool `Cpp2dpa` in combination with `1A1B1C` yields the first implementation of a decidability procedure for equivalence of protocols, for an unbounded number of sessions. However, the number of protocols covered so far is limited. A first reason yields in the limitations of the class of ping-pong protocols. However, another reason is the (too long) time needed to check for equivalence. Our transformation from protocols to automata using `Cpp2dpa` remains reasonably fast. Most of the execution time comes from `1A1B1C`. Since this tool is still in its early stage of development, we may hope for significant improvement of `1A1B1C`' performance in the next years.

REFERENCES

M. Arapinis, T. Chothia, E. Ritter, and M. Ryan. 2010. Analysing Unlinkability and Anonymity Using the Applied Pi Calculus. In *23rd Computer Security Foundations Symposium (CSF'10)*. IEEE Computer Society Press, 107–121.

D. Basin, S. Mödersheim, and L. Viganò. 2005. A symbolic model checker for security protocols. *Journal of Information Security* 4, 3 (2005), 181–208.

M. Baudet. 2005. Deciding Security of Protocols against Off-line Guessing Attacks. In *12th ACM Conference on Computer and Communications Security (CCS'05)*. ACM Press. DOI:http://dx.doi.org/10.1145/1102125

B. Blanchet. 2001. An efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *14th Computer Security Foundations Workshop (CSFW'01)*. IEEE Computer Society Press.

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

S. Boehm and S. Goeller. 2011. Language equivalence of deterministic real-time one-counter automata is NL-complete. In *Proceedings of the 36th International Symposium on Mathematical Foundations of Computer Science (MFCS'11) (LNCS)*, Vol. 6907. 194–205.

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

V. Cheval and B. Blanchet. 2013. Proving More Observational Equivalences with ProVerif. In *2nd Conference on Principles of Security and Trust (POST 2013) (Lecture Notes in Computer Science)*, David Basin and John Mitchell (Eds.), Vol. 7796. Springer, Rome, Italy, 226–246.

V. Cheval, H. Comon-Lundh, and S. Delaune. 2011. Trace Equivalence Decision: Negative Tests and Non-determinism. In *18th ACM Conference on Computer and Communications Security (CCS'11)*. ACM Press. http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/CCD-ccs11.pdf

Y. Chevalier and M. Rusinowitch. 2012. Decidability of Equivalence of Symbolic Derivations. *J. Autom. Reasoning* 48, 2 (2012), 263–292.

H. Comon-Lundh and V. Cortier. 2003. New Decidability Results for Fragments of First-Order Logic and Application to Cryptographic Protocols. In *14th International Conference on Rewriting Techniques and Applications (RTA'2003) (LNCS)*, Vol. 2706. Springer.

V. Cortier and S. Delaune. 2009. A method for proving observational equivalence. In *22nd IEEE Computer Security Foundations Symposium (CSF'09)*. IEEE Computer Society Press. DOI:http://dx.doi.org/10.1109/CSF.2009.9

C. Cremers. 2008. Unbounded verification, falsification, and characterization of security protocols by pattern refinement. In *15th ACM conference on Computer and communications security (CCS'08)*. ACM. DOI:http://dx.doi.org/10.1145/1455770.1455787

D. Denning and G. Sacco. 1981. Timestamps in Key Distribution Protocols. *Communication of the ACM* 24, 8 (1981), 533–536.

E. P. Friedman. 1976. The Inclusion Problem for Simple Languages. *Theor. Comput. Sci.* 1, 4 (1976), 297–316.

P. Henry and G. Sénizergues. 2013. LALBLC A Program Testing the Equivalence of dpda's. In *18th International Conference on Implementation and Application of Automata (CIAA 2013) (Lecture Notes in Computer Science)*, Vol. 7982. Springer, Halifax, NS, Canada, 169–180.

ICAO. 2008. *Machine Readable Travel Document*. Technical Report 9303. International Civil Aviation Organization.

M. Rusinowitch and M. Turuani. 2003. Protocol Insecurity with Finite Number of Sessions and Composed Keys is NP-complete. *Theoretical Computer Science* 299 (April 2003), 451–475. http://www.loria.fr/~rusi/pub/tcsprotocol.ps.gz

G. Sénizergues. 1997. The Equivalence Problem for Deterministic Pushdown Automata is Decidable. In *24th Int. Coll. on Automata, Languages and Programming (ICALP'97) (LNCS)*.

G. Sénizergues. 2001. L(A)=L(B)? Decidability results from complete formal systems. *Theor. Comput. Sci.* 251, 1-2 (2001), 1–166.

C. Stirling. 2002. Deciding DPDA Equivalence Is Primitive Recursive. In *29th International Colloquium on Automata, Languages and Programming ICALP'02 (LNCS)*. Springer.

A. Tiu and J. E. Dawson. 2010. Automating Open Bisimulation Checking for the Spi Calculus. In *23rd IEEE Computer Security Foundations Symposium (CSF'10)*. 307–321.

A. UNDECIDABILITY OF TRACE INCLUSION

The purpose of this section is to establish the following result.

THEOREM 3.6. *The following problem is undecidable.*

Input. *P and Q two protocols in $\mathcal{C}_{\mathsf{pp}}$.*
Output. *Whether P is trace included in Q, i.e. $P \sqsubseteq Q$.*

An instance of the PCP over the alphabet A is given by two sets of tiles $U = \{u_i \mid 1 \leq i \leq n\}$ and $V = \{v_i \mid 1 \leq i \leq n\}$ where $u_i, v_i \in A^*$. The problem consists of deciding whether there exists a non-empty sequence $i_1, \ldots, i_p$ over $\{1, \ldots, n\}$ such that $u_{i_1} \ldots u_{i_p} = v_{i_1} \ldots v_{i_p}$.

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. Conversely, if a solution to the Post Correspondence Problem does exit, it will lead the second process to react in a distinct way (by stopping its execution), breaking the trace inclusion property.

For each $i \in \{1, \ldots, n\}$, we define two (possibly empty) sets of words over A, namely $W_i \stackrel{\mathsf{def}}{=} A^{|v_i|} \smallsetminus \{v_i\}$, and $W'_i \stackrel{\mathsf{def}}{=} A^0 \cup A^1 \cup \ldots \cup A^{|v_i|-1}$ where $|v_i|$ denote the length of the word v_i.

Example A.1. Let $A = \{a, b\}$ and consider the following pairs of tiles (b, ϵ), (b, a), and (a, ba). This instance of PCP admits a solution. Indeed, the non-empty sequence 13 leads to the word $u_1 u_3 = v_1 v_3 = ba$. We have $W_1 = W'_1 = \emptyset$, $W_2 = \{b\}$ and $W'_2 = \{\epsilon\}$, and lastly $W_3 = \{aa, ab, bb\}$ and $W'_3 = \{a, b, \epsilon\}$.

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 denote by:

— $\overline{u}$ the term $\mathsf{senc}(\dots\mathsf{senc}(\epsilon, \alpha_1, z_1)\dots, \alpha_p, z_p)$; and
— $\overline{x.u}$ the term $\mathsf{senc}(\dots\mathsf{senc}(x, \alpha_1, z_1)\dots, \alpha_p, z_p)$

where $z_1, \dots, z_p$ are variables of sort rand. Note that if $u = \epsilon$ then $\overline{u} = \epsilon$, and $\overline{x.u} = x$.

Below, $\mathsf{k}_i, \mathsf{k}'_i$ with $i \in \{0, 1, 2, 3\}$ are constants in Σ_0 of sort SymKey, and for each $\alpha \in A$, we denote also by α its counterpart in Σ_0 (constants of sort SymKey). We denote ϵ a constant in Σ_0 of sort msg. These constants are initially unknown by the attacker and actually it is quite easy to see that they will be never revealed. Lastly, c, c_α, c_i, c' with $\alpha \in A$ and $i \in \{1, \dots, n\}$ are constant symbols of sort channel in $\mathcal{C}h$.

Let P_U and P_V be the following protocols.

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

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

The branch (start) is the only way to start an execution, then branches (1) are used to build a word $\alpha_1 \dots \alpha_n$ (that could be a Post word in case we consider a positive instance of PCP). This word will be represented through the term $\mathsf{senc}(\dots\mathsf{senc}(\epsilon, \alpha_1, r_1), \dots, \alpha_n, r_n)$ up to the choice of randoms. Then, branches (2) and (3) are used to unstack the different tiles $u_1, \dots, u_n$. Note that the purpose of having two similar branches (but using different keys) for this task is to ensure that we will unstack at least one tile, and thus the sequence $i_1 \dots i_p$ of indices is not empty. Then, reaching the empty word when unstacking these tiles will allow us to perform input/output on channel c' (branch (4)).

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

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

The protocol P_V has the same structure as P_U. However, it is more complex since we want P_V to follow the execution of P_U as soon as the execution does not correspond to a solution of the PCP problem. In particular, we do not want P_V to block in case it is not able to unstack a tile v_i. To achieve this, some additional branches are added (namely $(2'a)$ and $(2'b)$, as well as $(3'a)$ and $(3'b)$). Intuitively, those branches are triggered when $(2')$ and $(3')$ can not, and the resulting term is encrypted with a special key k_3' that will allow P_V to mimic the remaining of the execution using branch $(5')$. Now, regarding the branches on channel c', the idea is to allow P_V to mimic the behaviour of P_U only when the trace tr does not correspond to a solution of the PCP. To achieve this, we allow P_V to follow P_U only when the term given in input on channel c' is not a legal encoding of the empty word. Such a term will go through $(4'a)$ or $(4'b)$.

Note that, on both protocols, the terms that are outputted look like fresh random numbers due to fresh nonces occurring in every output and ignorance of the keys. In other words, the two frames resulting from the execution of respectively P_U and P_V always remain in static equivalence. Therefore, checking trace equivalence amounts into checking that any execution trace of P_U is a trace of P_V, and conversely.

LEMMA A.2. *The protocols P_U and P_V described above are in $\mathcal{C}_{\mathsf{pp}}$.*

PROOF. The only non-trivial point is to ensure that condition (2) stated in Definition 3.1 is satisfied, *i.e.* to ensure that pattern matching operated by inputs taking place on the same channel is exclusive. Regarding protocol P_U, when two inputs occur on the same channel c_i, we have that the outermost key is different. Regarding protocol P_V, the result also holds thanks to the exclusivity of the pattern matching obtained through a careful definition of sets W_i and W_i'. For instance, note that when $v_i = \epsilon$, $W_i = W_i' = \emptyset$, and thus there is no branch $(2'a)/(2'b)$ (resp. $(3'a)/(3'b)$). □

PROPOSITION A.3. *Let U/V be an instance of PCP. We have that $P_U \sqsubseteq P_V$ if, and only if, U/V is a negative instance of PCP (*i.e.* an instance with no solution).*

PROOF. We prove successively the two implications.

($\Rightarrow$) *If U/V is a positive instance of PCP then $P_U \not\sqsubseteq P_V$.* If U/V is a positive instance of PCP, there exists a non-empty sequence $i_1 \dots i_p$ over $\{1, \dots, n\}$ such that $u_{i_1} \cdot \ldots \cdot u_{i_p} = v_{i_1} \cdot \ldots \cdot v_{i_p}$.

Let $u = \alpha_1 \cdot \ldots \cdot \alpha_m$ be the resulting word over A. From this word and the sequence $i_1, \dots, i_p$, the attacker playing with P_u can build the term $\mathsf{senc}(\overline{u}, \mathsf{k}_0, r)$ representing the word u with branches (1) and then remove one by one the tiles u_{i_p} to u_{i_1} using (2) and (3). Let tr be the resulting trace of the protocol P_U:

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

where $\mathsf{io}(c, R, \mathsf{w}) \stackrel{\mathsf{def}}{=} \mathsf{in}(c, R).\mathsf{out}(c, \mathsf{w})$.

The trace tr models the fact that, given $\mathsf{senc}(\overline{u}, \mathsf{k}_0, r)$ (stored in w_{m+1}), 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 $\mathsf{senc}(\epsilon, \mathsf{k}_1, r)$ (stored in w_{m+p+1}) that can then be accepted as input on c'. In this execution, 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 this trace does exist in P_V, *i.e.* there exists no ψ such that $(\mathsf{tr}, \psi) \in \mathsf{trace}(P_V)$. Indeed, the pattern matching operated by P_V is exclusive once the term and the channel is fixed. Thus, P_V has no choice but to remove tiles v_{i_p} to v_{i_1} using $(2')$ and $(3')$ leading to the term $\mathsf{senc}(\epsilon, \mathsf{k}_1', r)$ (stored in w_{m+p+1}) as $\alpha_1 \dots \alpha_m$ 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 action $\mathsf{in}(c', \mathsf{w}_{m+p+1})$ will have no counterpart on P_V. So (tr, ϕ) has no equivalent trace in P_V, *i.e.* $P_U \not\sqsubseteq P_V$.

($\Leftarrow$) *If U/V is a negative instance of PCP then $P_U \sqsubseteq P_V$.* Let $(\mathsf{tr}, \phi) \in \mathsf{trace}(P_U)$, we aim at showing that there exists an equivalent trace $(\mathsf{tr}, \psi) \in \mathsf{trace}(P_V)$. Actually, since terms that are outputted by P_U and P_V look like fresh random numbers, we simply have to show that there exists ψ such that $(\mathsf{tr}, \psi) \in \mathsf{trace}(P_V)$. Two cases can occur for any trace $(\mathsf{tr}, \phi) \in \mathsf{trace}(P_U)$:

— tr contains no input on channel c'. In such a case, by construction of P_V, the frame ψ can be built by following the sequence of channels used in tr and choosing the adequate filtering. 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. Note that when the term given in input is of the form $\mathsf{senc}(\epsilon, \mathsf{k}'_3, r)$ for some r, it would be accepted on any channel.
— tr contains an input on channel c'. In such a case, this means that the associated term $\mathsf{senc}(\overline{\alpha_1 \dots \alpha_m}, \mathsf{k}_0, r)$ that has been built using channels c_α with $\alpha \in A$ is a word made of tiles in $\{u_1, \dots, u_n\}$. 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 this particular instance of PCP has no solution, such a word $\alpha_1 \dots \alpha_m$ cannot be a Post word and thus it cannot be decomposed using tiles in $\{v_1, \dots, v_n\}$ following the same sequence of indices: because the filtering in P_V is also exhaustive, messages outputted by P_V from a certain point will be either encrypted by k'_3 or will reach the end of the sequence with a term of the form $\mathsf{senc}(u, \mathsf{k}'_1, r)$ with u different from the constant ϵ. Thanks to branches $(4'a)$, $(4'b)$, and $(5')$, P_V will be able to follow P_U.

Hence, for any trace $(\mathsf{tr}, \phi) \in \mathsf{trace}(P_U)$ there exists a trace $(\mathsf{tr}, \psi) \in \mathsf{trace}(P_V)$. It remains to show that $\phi \sim \psi$. This is due to the fact that both ϕ and ψ are of the form $\{\mathsf{w}_1 \triangleright \mathsf{senc}(m_1, k_1, r_1), \dots, \mathsf{w}_n \triangleright \mathsf{senc}(m_n, k_n, r_n)\}$ where the k_i are non deducible and the r_i are "fresh" in the sense that they are all distinct and non deducible. We therefore conclude that $P_U \sqsubseteq P_V$. $\square$

Theorem 3.6 directly follows from Proposition A.3 and the undecidability of the Post Correspondence Problem.

B. GETTING RID OF THE ATTACKER

LEMMA B.1. *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 $\mathsf{s} \in \{\mathsf{SymKey}, \mathsf{PubKey}, \mathsf{PrivKey}\}$.

— *$R\phi\downarrow = \mathsf{k}$ if, and only if, $R\psi\downarrow = \alpha(\mathsf{k})$;*
— *$R\phi\downarrow = \mathsf{k}^{-1}$ if, and only if, $R\psi\downarrow = (\alpha(\mathsf{k}))^{-1}$;*

and conversely, for every $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$ there exists a trace $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ satisfying the same properties.

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 recipe R such that $R\phi\downarrow = \mathsf{k}$, we define $\alpha(\mathsf{k}) = R\psi\downarrow$ where ψ is the only frame such that $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$.

The existence of such a frame comes from the fact that $P \approx Q$, whereas its 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:

— *$R\psi\downarrow$ is a constant of sort* s. We have that there exists a trace $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ such that $R\phi\downarrow = \mathsf{k} \in K_P$. Since $P \approx Q$ and Q is in $\mathcal{C}_{\mathsf{pp}}$, we consider the trace $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$. By definition of static equivalence, we have that $R\psi\downarrow$ is a constant of sort s. Otherwise, we would have that $\mathsf{senc}(\mathsf{start}, R, r_i)\phi \in \mathcal{T}(\Sigma, \mathcal{N})$ whereas $\mathsf{senc}(\mathsf{start}, R, r_i)\psi \notin \mathcal{T}(\Sigma, \mathcal{N})$ if $\mathsf{s} = \mathsf{SymKey}$ (the resulting term is not properly sorted). The same argument applies with aenc and sign for s equal to PubKey and $\mathsf{PrivKey}$ respectively.

— *We have that* $|\mathcal{K}_P| = |\mathcal{K}_Q|$. Suppose *ad absurdum* that, for instance, $|\mathcal{K}_P| < |\mathcal{K}_Q|$. Since every element of $\mathcal{K}_Q$ is deducible (and due to the shape of the protocols under study), there exists $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$ such that for all $\mathsf{k} \in \mathcal{K}_Q$, there exists a recipe R_k such that $R_\mathsf{k}\psi\downarrow = \mathsf{k}$. In particular, when $k \neq k'$, we have that $R_k\psi\downarrow \neq R_{k'}\psi\downarrow$. Since $P \approx Q$, there exists a frame ϕ such that $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$. Thanks to previous item, we know that $R_k\phi\downarrow$ (resp $R_{k'}\phi\downarrow$) has the same sort as $R_k\psi\downarrow$ (resp. $R_{k'}\psi\downarrow$), *i.e.* sort key. As $|\mathcal{K}_P| < |\mathcal{K}_Q|$, there exist two distinct keys k and k' such that $R_\mathsf{k}\phi\downarrow = R_{\mathsf{k}'}\phi\downarrow$. Hence ϕ and ψ are not statically equivalent, contradicting the fact that $P \approx Q$. The case where $|\mathcal{K}_Q| < |\mathcal{K}_P|$ can be handle similarly.

— *α is a function.* Suppose there exist a trace $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$, a recipe R_i and a corresponding equivalence trace $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$ such that $R_i\phi\downarrow = \mathsf{k}$ and $R_i\psi\downarrow = \mathsf{k}'$; a trace $(\mathsf{tr}', \phi') \in \mathsf{trace}(P)$, a recipe R_j and a corresponding equivalence trace $(\mathsf{tr}', \psi') \in \mathsf{trace}(Q)$ such that $R_j\phi'\downarrow = \mathsf{k}$ but $R_j\psi'\downarrow = \mathsf{k}''$ with $\mathsf{k}' \neq \mathsf{k}''$. Considering the trace made up of the trace tr followed by tr', it is then possible to exhibit a witness of non-equivalence. More precisely, relying on R_i and R_j we can build a test that holds in the resulting frame when executing P, whereas this test will not hold on the frame resulting from the execution of Q.

Now we show that α is an injection, *i.e.* $\alpha(\mathsf{k}) \neq \alpha(\mathsf{k}')$ as soon as k, k' are two distinct elements of $\mathcal{K}_P$. Suppose, as previously, there exist a trace $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$, a recipe R_i and a corresponding equivalence trace $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$ such that $R_i\phi\downarrow = \mathsf{k}$ and $R_i\psi\downarrow = \alpha(\mathsf{k})$; a trace $(\mathsf{tr}', \phi') \in \mathsf{trace}(P)$, a recipe R_j and a corresponding equivalence trace $(\mathsf{tr}', \psi') \in \mathsf{trace}(Q)$ such that $R_j\phi'\downarrow = \mathsf{k}'$ but $R_j\psi'\downarrow = \alpha(\mathsf{k})$ with $\mathsf{k} \neq \mathsf{k}'$. Considering the trace made up of the trace tr followed by tr', it is then possible to exhibit a witness of non-equivalence. More precisely, relying on R_i and R_j we can build a test that holds in the frame resulting from the execution of P and that does not hold when executing Q. Thus, we have now prove that α is a bijection.

Note that we have already proved that: $R\phi\downarrow = \mathsf{k}$ if, and only, if $R\psi\downarrow = \alpha(\mathsf{k})$.

To show that α satisfies the last condition (item 3), suppose that $\mathsf{k} \in \mathcal{K}_P$, and $R\phi\downarrow = \mathsf{k}^{-1}$. 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 SymKey, 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 $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ and a recipe R' such that $R'\phi\downarrow = \mathsf{k} \in K_P$. Since $P \approx Q$, consider the corresponding equivalence trace $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$. Consider the recipes $R_1 = \mathsf{aenc}(\mathsf{start}, R', n)$ 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, $R\psi\downarrow = (R'\psi)^{-1}$. As we have already proved that α preserves sorts, we get that $R_2\psi\downarrow$ is of sort msg if, and only if, $\alpha(\mathsf{k}^{-1}) = R\psi\downarrow = (R'\psi\downarrow)^{-1} = (\alpha(\mathsf{k}))^{-1}$. Hence α is compatible with the inverse function. The same argument can be used if k is of sort $\mathsf{PrivKey}$ 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 recipe R such that $R\phi\downarrow = \mathsf{k}$, $\alpha(\mathsf{k}) = R\psi\downarrow$. But as α' satisfy a similar property, we get that $R\psi\downarrow = \alpha'(\mathsf{k})$, contradicting our hypothesis. Hence α is unique. Determinism of P and Q then ensures that traces of P and Q are uniquely matched (as $P \approx Q$), thus guaranteeing the converse part of the Lemma. □

LEMMA B.2. *Let P and Q be two protocols in $\mathcal{C}_{\mathsf{pp}}$ respectively disclosing two sets of keys K and K' as in Lemma 4.12. Then $P \approx Q$ if, and only if, $\bar{P} \approx_{\mathsf{fwd}} \bar{Q}$ where $\bar{P}$ and $\bar{Q}$ are as defined in Lemma 4.14. We call $\mathcal{T}_{\mathsf{oracle}}$ the transformation taking a pair of protocols (P, Q) satisfying the aforementioned condition and returning the pair $(\bar{P}, \bar{Q})$ presently defined.*

PROOF. Let $\mathcal{K}_P$ (resp. $\mathcal{K}_Q$) be the set of deducible constants of sort key that occur in P (resp. Q). We recall, that, as a consequence of Lemma 4.12, 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 $\bar{Q}$. In the following, we will assume that 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.

For sake of clarity of the construction explained below, we actually show that:

$$\bar{P} \approx_{\mathsf{fwd}} \bar{Q} \text{ if, and only if } P^+ \approx Q^+$$

where $P^+ = P \mid !\mathsf{in}(c, x).\mathsf{out}(c, x)$ and $Q^+ = Q \mid !\mathsf{in}(c, x).\mathsf{out}(c, x)$ for some fresh channel name c. Then, it is easy to conclude at the expected result relying on the fact that $P \approx Q$ is equivalent to $P^+ \approx Q^+$.

($\Rightarrow$) First, suppose $\bar{P} \not\approx_{\mathsf{fwd}} \bar{Q}$. Assume that there exists $(\mathsf{tr}, \phi) \in \mathsf{trace}_{\mathsf{fwd}}(\bar{P})$ such that there is no equivalent frame ψ such that $(\mathsf{tr}, \psi) \in \mathsf{trace}_{\mathsf{fwd}}(\bar{Q})$. We define $(\mathsf{tr}', \phi) \in \mathsf{trace}(P^+)$ as follows:

— every sequence $\mathsf{in}(c^{\mathsf{senc}}_{\mathsf{k},\alpha(\mathsf{k})}, R).\mathsf{out}(c^{\mathsf{senc}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{w}')$ in tr is replaced by the sequence $\mathsf{in}(c, \mathsf{senc}(R, \mathsf{w}_k^0, n)).\mathsf{out}(c, \mathsf{w}')$ in tr' where n is a fresh name.
— every sequence $\mathsf{in}(c^{\mathsf{sdec}}_{\mathsf{k},\alpha(\mathsf{k})}, R).\mathsf{out}(c^{\mathsf{sdec}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{w}')$ in tr is replaced by the sequence $\mathsf{in}(c, \mathsf{sdec}(R, \mathsf{w}_k^0)).\mathsf{out}(c, \mathsf{w}')$ in tr'.
— every sequence $\mathsf{in}(c^{\mathsf{aenc}}_{\mathsf{k},\alpha(\mathsf{k})}, R).\mathsf{out}(c^{\mathsf{aenc}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{w}')$ in tr is replaced by the sequence $\mathsf{in}(c, \mathsf{aenc}(R, \mathsf{w}_k^0, n)).\mathsf{out}(c, \mathsf{w}')$ in tr' where n is a fresh name.
— every sequence $\mathsf{in}(c^{\mathsf{adec}}_{\mathsf{k},\alpha(\mathsf{k})}, R).\mathsf{out}(c^{\mathsf{adec}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{w}')$ in tr is replaced by the sequence $\mathsf{in}(c, \mathsf{adec}(R, \mathsf{w}_k^0)).\mathsf{out}(c, \mathsf{w}')$ in tr'.
— every sequence $\mathsf{in}(c^{\mathsf{sign}}_{\mathsf{k},\alpha(\mathsf{k})}, R).\mathsf{out}(c^{\mathsf{sign}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{w}')$ in tr is replaced by the sequence $\mathsf{in}(c, \mathsf{sign}(R, \mathsf{w}_k^0, n)).\mathsf{out}(c, \mathsf{w}')$ in tr' where n is a fresh name.
— every sequence $\mathsf{in}(c^{\mathsf{check}}_{\mathsf{k},\alpha(\mathsf{k})}, R).\mathsf{out}(c^{\mathsf{check}}_{\mathsf{k},\alpha(\mathsf{k})}, \mathsf{w}')$ in tr is replaced by the sequence $\mathsf{in}(c, \mathsf{check}(R, \mathsf{w}_k^0)).\mathsf{out}(c, \mathsf{w}')$ in tr'.

Note that by definition of a trace being in $\mathsf{trace}_{\mathsf{fwd}}(\bar{P})$, we have that R is either a variable w or the constant start. 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 $\mathsf{trace}_{\mathsf{fwd}}(\bar{Q})$ and be equivalent to (tr, ϕ).

($\Leftarrow$) Now, suppose $P \not\approx Q$. We have that $P^+ \not\approx Q^+$, and we can even assume that $P^+ \not\approx^{\mathsf{io}*} Q^+$. We consider a witness of this non-equivalence, *i.e.* a trace tr such that $(\mathsf{tr}, \phi) \in \mathsf{trace}^{\mathsf{io}*}(P^+)$ and for which there exists no equivalent frame ψ such that $(\mathsf{tr}, \psi) \in \mathsf{trace}^{\mathsf{io}*}(Q^+)$. Actually, we can even assume w.l.o.g. that:

— every input recipe in tr on a channel different from c is either a variable w or the constant start;
— every input recipe in tr on channel c involves at most one function symbol in Σ_{pub};
— $\phi \not\sim_{\mathsf{fwd}} \psi$, *i.e.* we consider recipes that are either variables or the constant start.

We consider the shortest trace $(\mathsf{tr}, \phi) \in \mathsf{trace}^{\mathsf{io}*}(P)$, in terms of number of transitions, such that there is no equivalent frame ψ satisfying $(\mathsf{tr}, \psi) \in \mathsf{trace}^{\mathsf{io}*}(Q)$, and for which all the requirements listed above are satisfied.

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.

First, if we consider the case where there exists no ψ such that $(\mathsf{tr}, \psi) \notin \mathsf{trace}^{\mathsf{io}*}(Q)$. Because random seeds are not filtered in protocols of $\mathcal{C}_{\mathsf{pp}}$ (every input pattern contains distinct variables as third argument), we can rename some occurrences of the random seeds of the attacker (*i.e.* the random seeds appearing in the recipes on channel c) by fresh random seeds without changing the status of the trace (*i.e.* the fact that the trace is executable or not). Given tr_ρ such a trace obtained by renaming, we have that $(\mathsf{tr}_\rho, \phi_\rho) \in \mathsf{trace}^{\mathsf{io}*}(P)$ for some frame ϕ_ρ whereas $(\mathsf{tr}_\rho, \psi_\rho) \notin \mathsf{trace}^{\mathsf{io}*}(Q)$ for any frame ψ_ρ. And it particular, if we choose tr_ρ such that there are no two identical nonces in its image, we get a witness of non-equivalence with pairwise distinct random seeds for the attacker.

Now, we consider the case where $(\mathsf{tr}, \psi) \in \mathsf{trace}^{\mathsf{io}*}(Q)$ but $\phi \not\sim_{\mathsf{fwd}} \psi$. Suppose r is a random seed which appears twice in tr, in two contexts $\mathsf{f}(\mathsf{w}_i, \mathsf{w}_j, r)$ and $\mathsf{f}(\mathsf{w}'_i, \mathsf{w}'_j, r)$ for some $\mathsf{f} \in \Sigma_{\mathsf{pub}}$ with $\mathsf{w}_i\phi = \mathsf{w}'_i\phi$ and $\mathsf{w}_j\phi = \mathsf{w}'_j\phi$. Because tr is a minimal witness of non-equivalence, $\phi_{-1} \sim_{\mathsf{fwd}} \psi_{-1}$ where ϕ_{-1} (resp. ψ_{-1}) denotes ϕ (resp. ψ) minus its last element. Consequently we also have that $\mathsf{w}_i\psi = \mathsf{w}'_i\psi$ and $\mathsf{w}_j\psi = \mathsf{w}'_j\psi$, as $\mathsf{w}_i, \mathsf{w}_j, \mathsf{w}'_i, \mathsf{w}'_j \in dom(\phi_{-1})$ (they are used in input recipes). Let w and w' be the corresponding outputs of the recipes $\mathsf{f}(\mathsf{w}_i, \mathsf{w}_j, r)$ and $\mathsf{f}(\mathsf{w}'_i, \mathsf{w}'_j, r)$ and assume w appears before w' in tr: we now have that $\mathsf{w} = \mathsf{w}'$ in both ϕ and ψ, and we can safely replace any occurrence of w' in tr by w. The resulting trace is still a witness of non-equivalence as the substitution replace identical terms in ψ.

Thus, it remains only to consider the case where a random seed appears twice in tr but such that either the function symbol, the plaintext or the keys are different. Formally, consider the two contexts $\mathsf{f}(\mathsf{w}_i, \mathsf{w}_j, r)$ and $\mathsf{g}(\mathsf{w}'_i, \mathsf{w}'_j, r)$ with $\mathsf{f}, \mathsf{g} \in \Sigma_{\mathsf{pub}}$, w and w' their respective outputs variables as before; and either $\mathsf{w}_i\phi \neq \mathsf{w}'_i\phi$, $\mathsf{w}_j\phi \neq \mathsf{w}'_j\phi$ or $\mathsf{f} \neq \mathsf{g}$. Following the same reasoning as before, as $\phi_{-1} \sim_{\mathsf{fwd}} \psi_{-1}$, the same inequality has to hold in ψ. Consider the test $\mathsf{w}_k = \mathsf{w}'_k$ which distinguishes between ϕ and ψ: suppose $\mathsf{w}_k\phi = \mathsf{w}'_k\phi$ but $\mathsf{w}_k\psi \neq \mathsf{w}'_k\psi$. Replacing r by r' in $\mathsf{g}(\mathsf{w}'_i, \mathsf{w}'_j, r)$ will still lead to $\mathsf{w}_k\psi \neq \mathsf{w}'_k\psi$ (after replacement) as no equality between subterms is added. But if $\mathsf{w}_k\phi \neq \mathsf{w}'_k\phi$ (after replacement), it would imply that there were two subterms which became different, and were identical before: but, because the first step already took care of recipes introducing the same random seed twice in the same context, and the protocols in $\mathcal{C}_{\mathsf{pp}}$ cannot use a random seed from an input to use it in another encryption, it is impossible.

Hence, 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 $\mathsf{f}(u, v, w)$ in (tr, ϕ) are distinct.

Let us now define a corresponding trace $(\bar{\mathsf{tr}}, \bar{\phi}) \in \mathsf{trace}_{\mathsf{fwd}}(\bar{P})$.

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

Note that each recipe R and R_k above is a variable w or the constant start. The corresponding frame $\bar{\phi}$ is then defined according to our semantics. Since we have assume that the random seed occurring in the recipes in tr are distinct, we have that $\bar{\phi} = \phi$.

Finally, because $(\mathsf{tr}, \phi) \in \mathsf{trace}^{\mathsf{io}*}(P)$ 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, $(\bar{\mathsf{tr}}, \bar{\phi}) \in \mathsf{trace}_{\mathsf{fwd}}(\bar{P})$ has no equivalent in $\bar{Q}$. □

C. ENCODING A PROTOCOL INTO A REAL-TIME GPDA

C.1. Characterization of trace equivalence

LEMMA C.1. *Let P and Q be two procotols in $\mathcal{C}_{\mathsf{pp}}$, if $P \approx_{\mathsf{fwd}} Q$ then for every trace $(\mathsf{tr}, \sigma_P) \in \mathsf{trace}_{\mathsf{fwd}}(P)$ and every $\mathsf{w}, \mathsf{w}' \in dom(\sigma_P)$, if $\mathsf{w}\sigma_P = \mathsf{w}'\sigma_P = \mathsf{c}$ for some constant c, then $\mathsf{w}\sigma_Q = \mathsf{w}'\sigma_Q = \mathsf{c}'$ for some constant c' where σ_Q is the frame such that $(\mathsf{tr}, \sigma_Q) \in \mathsf{trace}(Q)$.*

PROOF. First, note that the frame σ_Q mentionned in the lemma is unique up to some alpha-renaming of the randoms that occur in σ_P. Thus, the choice of the frame σ_Q does not change anything regarding the result that we want to prove.

Actually, the only non-trivial point to prove is that if $\mathsf{w}\sigma_P = \mathsf{c}$, then $\mathsf{w}\sigma_Q$ is necessarily a constant too. Since protocols in $\mathcal{C}_{\mathsf{pp}}$ have a replication for every branch, consider the trace obtained by "playing twice" the trace tr in P and Q, *i.e.* given $(\mathsf{tr}, \sigma_P) \in \mathsf{trace}_{\mathsf{fwd}}(P)$ with

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

build $(\mathsf{tr}', \sigma'_P) \in \mathsf{trace}_{\mathsf{fwd}}(P)$ where:

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

where every occurrence of start in tr is kept in $\bar{\mathsf{tr}}$ but occurrences of w_k are replaced by $\mathsf{w}_{|\sigma_P|+k}$, $|\sigma_P|$ being the cardinal of $dom(\sigma_P)$; and $\mathsf{tr}.\bar{\mathsf{tr}}$ denotes the concatenation of the two sequences of labels, which is a valid trace, *i.e.* $(\mathsf{tr}', \sigma'_P) \in \mathsf{trace}_{\mathsf{fwd}}(P)$. We get symmetrically $(\mathsf{tr}', \sigma'_Q) \in \mathsf{trace}_{\mathsf{fwd}}(Q)$. In particular, there exists $\mathsf{w}_* \in dom(\sigma'_P)$ with $l < *$ such that $\mathsf{w}\sigma'_P = \mathsf{w}_*\sigma'_P = \mathsf{c}$ and the test $\mathsf{w} = \mathsf{w}_*$ is disjoint, *i.e.* $\mathsf{seq}_{\mathsf{tr}'}(\mathsf{w})$ and

$\mathsf{seq}_{\mathsf{tr}'}(\mathsf{w}_*)$ share no common prefix. As $P \approx_{\mathsf{fwd}} Q$, necessarily $\mathsf{w}\sigma'_Q = \mathsf{w}_*\sigma'_Q$. Now, because the test is disjoint, $\mathsf{w}\sigma'_Q$ and $\mathsf{w}_*\sigma'_Q$ could not share any random nonces. Hence, $\mathsf{w}\sigma_Q$ is a constant. □

LEMMA C.2. *Let P and Q be two procotols in $\mathcal{C}_{\mathsf{pp}}$ such that $P \approx_{\mathsf{fwd}} Q$. For every trace $(\mathsf{tr}, \sigma_P) \in \mathsf{trace}_{\mathsf{fwd}}(P)$, every $\mathsf{w}, \mathsf{w}' \in dom(\sigma_P)$ such that the test $\mathsf{w} = \mathsf{w}'$ is σ_P-valid, σ_P-guarded, and pulled-up in (tr, σ_P), we have that $\mathsf{w} = \mathsf{w}'$ is σ_Q-valid, σ_Q-guarded, and pulled-up in (tr, σ_Q) where σ_Q is the frame such that $(\mathsf{tr}, \sigma_Q) \in \mathsf{trace}_{\mathsf{fwd}}(Q)$*

PROOF. First, note that the frame σ_Q mentionned in the lemma is unique up to some alpha-renaming of the randoms that occur in σ_P. Thus, the choice of the frame σ_Q does not change anything regarding the result that we want to prove.

The only non-trivial point to prove is that if the test $\mathsf{w} = \mathsf{w}'$ is σ_P-valid, σ_P-guarded, and pulled-up in (tr, σ_P) then it is also σ_Q-guarded and pulled-up in (tr, σ_Q). Note that it is necessarily σ_Q-valid since $P \approx_{\mathsf{fwd}} Q$. Actually, we can still assume that the test $\mathsf{w} = \mathsf{w}'$ is σ_Q-guarded (it would otherwise contradict Lemma C.1).

Let $\mathsf{pref} = \mathsf{io}(c_{i_0}, \mathsf{start}, \mathsf{w}_{j_0}) \dots \mathsf{io}(c_{i_p}, \mathsf{w}_{j_{p-1}}, \mathsf{w}_{j_p})$ be the maximal common prefix of $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w})$ and $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w}')$. Now, it remains to show that $\mathsf{w} = \mathsf{w}'$ is pulled-up in (tr, σ_Q), *i.e.* $\mathsf{w}\sigma_Q$ does not occur as a subterm in $\mathsf{w}_{j_{-1}}\sigma_Q, \mathsf{w}_{j_0}\sigma_Q, \dots \mathsf{w}_{j_{p-1}}\sigma_Q$ where $\mathsf{w}_{j_{-1}}\sigma_Q = \mathsf{start}$.

Assume that this is not the case, we will show that there exists a trace $(\mathsf{tr}^*, \sigma_Q^*) \in \mathsf{trace}_{\mathsf{fwd}}(Q)$, $\mathsf{w}_*, \mathsf{w}'_* \in dom(\sigma_Q^*)$ such that $\mathsf{w}_*\sigma_Q^* = \mathsf{w}'_*\sigma_Q^*$ whereas $\mathsf{w}_*\sigma_P^* \neq \mathsf{w}'_*\sigma_P^*$, where σ_P^* is the frame such that $(\mathsf{tr}^*, \sigma_P^*) \in \mathsf{trace}_{\mathsf{fwd}}(P)$. Note that such a frame necessarily exists since otherwise it trivially contradicts our hyptothesis.

Let $p' \in \{0, \dots, p-1\}$ be the smallest indice such that $\mathsf{w}\sigma_Q$ occurs as a subterm in $\mathsf{w}_{j_{p'}}\sigma_Q$. We have that:

$$\mathsf{pref} = s_1.\mathsf{io}(c_{i_{p'}}, \mathsf{w}_{j_{p'-1}}, \mathsf{w}_{j_{p'}}).s_2 \quad \text{and} \quad \begin{cases} \mathsf{seq}_{\mathsf{tr}}(\mathsf{w}) = \mathsf{pref}.s_3 \\ \mathsf{seq}_{\mathsf{tr}}(\mathsf{w}') = \mathsf{pref}.s'_3 \end{cases}$$

for some sequence s_1, s_2, s_3, and s'_3.

From these sequences we can define $(\mathsf{tr}^*, \sigma_Q^*)$ with $\mathsf{tr}^* = \mathsf{tr}.\bar{\mathsf{tr}}$. Intuitively, the trace $\bar{\mathsf{tr}}$ is obtained relying on the sequence of channels as indicated in the sequence $s_2.s'_3$ using systematically the last generated recipe to feed the following input, and $\mathsf{w}_{j_{p'}}$ to start. More precisely, assuming that

$$\mathsf{seq}_{\mathsf{tr}}(\mathsf{w}') = s_1.\mathsf{io}(c_{i_{p'}}, \mathsf{w}_{j_{p'-1}}, \mathsf{w}_{j_{p'}}).\mathsf{io}(c_{k_1}, \mathsf{w}_{j_{p'}}, \mathsf{w}_{l_1}).\mathsf{io}(c_{k_2}, \mathsf{w}_{l_1}, \mathsf{w}_{l_2}) \dots \mathsf{io}(c_{k_\ell}, \mathsf{w}_{l_{\ell-1}}, \mathsf{w}_{l_\ell})$$

we have that:

$$\bar{\mathsf{tr}} = \mathsf{io}(c_{k_1}, \mathsf{w}_{j_{p'}}, \mathsf{w}_{|\sigma_P|+1}).\mathsf{io}(c_{k_2}, \mathsf{w}_{|\sigma_P|+1}, \mathsf{w}_{|\sigma_P|+2}) \dots \mathsf{io}(c_{k_\ell}, \mathsf{w}_{|\sigma_P|+l-1}, \mathsf{w}_{|\sigma_P|+\ell})$$

and σ_Q^* defined as expected relying on our semantics. Let $\mathsf{w}_* = \mathsf{w}$ and $\mathsf{w}'_* = \mathsf{w}_{|\sigma_P|+\ell}$. We can now show that:

(1) *The test $\mathsf{w}_* = \mathsf{w}'_*$ is σ_Q^*-valid and σ_Q^*-guarded.* Indeed, by definition of tr^*, $\mathsf{w}'_*\sigma_Q^*$ and $\mathsf{w}'\sigma_Q$ are already equal up to a renaming of random seeds, as the channel components of $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w}')$ and $\mathsf{seq}_{\mathsf{tr}_*}(\mathsf{w}'_*)$ match. As $\mathsf{w}_*\sigma_Q^* = \mathsf{w}\sigma_Q = \mathsf{w}'\sigma_Q$, $\mathsf{w}_*\sigma_Q^*$ and $\mathsf{w}'_*\sigma_Q^*$ are equal up to a renaming of their random seeds. Lastly, we have that $\mathsf{w}_*\sigma_Q^*$ and $\mathsf{w}'_*\sigma_Q^*$ are both subterms of $\mathsf{w}_{j_{p'}}\sigma_Q^*$, hence $\mathsf{w}_*\sigma_Q^* = \mathsf{w}'_*\sigma_Q^*$.
(2) *The test $\mathsf{w}_* = \mathsf{w}'_*$ is pulled-up in $(\mathsf{tr}^*, \sigma_Q^*)$.* This is by construction of tr^*.

Finally, as $P \approx_{\mathsf{fwd}} Q$, there exists σ_P^* such that $(\mathsf{tr}^*, \sigma_P^*) \in \mathsf{trace}_{\mathsf{fwd}}(P)$. But now $\mathsf{w}_* = \mathsf{w}'_*$ is σ_Q^*-valid, σ_Q^*-guarded and pulled-up in $(\mathsf{tr}^*, \sigma_Q^*)$. Moreover, we are now in a situation where the top-level random seeds of $\mathsf{w}_*\sigma_P^*$ and $\mathsf{w}'_*\sigma_P^*$ are generated outside the common prefix of $\mathsf{seq}_{\mathsf{tr}^*}(\mathsf{w}_*)$ and $\mathsf{seq}_{\mathsf{tr}^*}(\mathsf{w}'_*)$, and thus it implies that $\mathsf{w}_*\sigma_P^* \neq \mathsf{w}'_*\sigma_P^*$, contradicting the equivalence $P \approx_{\mathsf{fwd}} Q$. □

LEMMA C.3. *Let P and Q be two protocols in $\mathcal{C}_{\mathsf{pp}}$, then $P \approx_{\mathsf{fwd}} Q$ if, and only if, the following four conditions are satisfied:*

— CONST$_P$*: For all* $(\mathsf{tr}, \sigma_P) \in \mathsf{trace}_{\mathsf{fwd}}(P)$*, there exists a frame* σ_Q *such that* $(\mathsf{tr}, \sigma_Q) \in \mathsf{trace}_{\mathsf{fwd}}(Q)$ *and for every* $\mathsf{w}, \mathsf{w}' \in dom(\sigma_P)$ *and for every constant* $\mathsf{c} \in \Sigma_0 \cup \{\mathsf{start}\}$, $\mathsf{w}\sigma_P = \mathsf{w}'\sigma_Q = \mathsf{c}$ *if, and only if, there exists a constant* $\mathsf{c}' \in \Sigma_0 \cup \{\mathsf{start}\}$ *such that* $\mathsf{w}\sigma_Q = \mathsf{w}'\sigma_Q = \mathsf{c}'$.
— CONST$_Q$*: Similarly swapping the roles of P and Q.*

— GUARDED$_P$*: For all* $(\mathsf{tr}, \sigma_P) \in \mathsf{trace}_{\mathsf{fwd}}(P)$*, there exists a frame* σ_Q *such that* $(\mathsf{tr}, \sigma_Q) \in \mathsf{trace}_{\mathsf{fwd}}(Q)$ *and every test that is* σ_P*-valid,* σ_P*-guarded, and pulled-up in* (tr, σ_P) *is also* σ_Q*-valid,* σ_Q*-guarded, and pulled-up in* (tr, σ_Q).
— GUARDED$_Q$*: Similarly swapping the roles of P and Q.*

PROOF. We prove the two directions separately.
($\Rightarrow$) This implication is a direct consequence of Lemma C.1 and Lemma C.2.

($\Leftarrow$) Suppose that $P \not\approx_{\mathsf{fwd}} Q$. This means that there exists for instance $(\mathsf{tr}, \sigma_P) \in \mathsf{trace}_{\mathsf{fwd}}(P)$ such that either there exists no frame σ_Q such that $(\mathsf{tr}, \sigma_Q) \in \mathsf{trace}_{\mathsf{fwd}}(Q)$, in which case conditions CONST$_P$ and GUARDED$_P$ fail, or σ_Q is indeed defined and there exists a test $\mathsf{w} = \mathsf{w}'$ such that $\mathsf{w}\sigma_P = \mathsf{w}'\sigma_P$ but $\mathsf{w}\sigma_Q \neq \mathsf{w}'\sigma_Q$ (or the converse). Let us assume that $\mathsf{w}\sigma_P = \mathsf{w}'\sigma_P$ but $\mathsf{w}\sigma_Q \neq \mathsf{w}'\sigma_Q$.

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

Otherwise, we have that the test $\mathsf{w} = \mathsf{w}'$ is σ_P-valid and σ_P-guarded. From tr and $\mathsf{w} = \mathsf{w}'$, we will build a new trace $(\mathsf{tr}^*, \sigma_P^*)$ and a new test $\mathsf{w}_* = \mathsf{w}'_*$ which is σ_P^*-valid, σ_P^*-guarded, and also pulled-up in $(\mathsf{tr}^*, \sigma_P^*)$. Actually, we proceed as in the proof of the previous lemma.

Let $\mathsf{pref} = \mathsf{io}(c_{i_0}, \mathsf{start}, \mathsf{w}_{j_0}) \ldots \mathsf{io}(c_{i_p}, \mathsf{w}_{j_{p-1}}, \mathsf{w}_{j_p})$ be the maximal common prefix of $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w})$ and $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w}')$. Let $p' \in \{0, \ldots, p-1\}$ be the smallest indice such that $\mathsf{w}\sigma_P$ occurs as a subterm in $\mathsf{w}_{j_{p'}}\sigma_P$. Note that if this indice does not exist then the test $\mathsf{w} = \mathsf{w}'$ is already pulled-up in (tr, σ_P) and we are done.

We have that:

$$\mathsf{pref} = s_1.\mathsf{io}(c_{i_{p'}}, \mathsf{w}_{j_{p'-1}}, \mathsf{w}_{j_{p'}}).s_2 \quad \text{and} \quad \begin{cases} \mathsf{seq}_{\mathsf{tr}}(\mathsf{w}) \;=\; \mathsf{pref}.s_3 \\ \mathsf{seq}_{\mathsf{tr}}(\mathsf{w}') \;=\; \mathsf{pref}.s'_3 \end{cases}$$

for some sequence s_1, s_2, s_3, and s'_3.

From these sequences, we can define $(\mathsf{tr}^*, \sigma_P^*)$ with $\mathsf{tr}^* = \mathsf{tr}, \bar{\mathsf{tr}}$. Intuitively, the trace $\bar{\mathsf{tr}}$ is obtained relying on the sequence of channels as indicated in the sequence $s_2.s'_3$ using systematically the last generated recipe to feed the following input, and $\mathsf{w}_{j_{p'}}$ to start. More precisely, assuming that

$$\mathsf{seq}_{\mathsf{tr}}(\mathsf{w}') = s_1.\mathsf{io}(c_{i_{p'}}, \mathsf{w}_{j_{p'-1}}, \mathsf{w}_{j_{p'}}).\mathsf{io}(c_{k_1}, \mathsf{w}_{j_{p'}}, \mathsf{w}_{l_1}).\mathsf{io}(c_{k_2}, \mathsf{w}_{l_1}, \mathsf{w}_{l_2}) \ldots \mathsf{io}(c_{k_\ell}, \mathsf{w}_{l_{\ell-1}}, \mathsf{w}_{l_\ell})$$

we have that:

$$\bar{\mathsf{tr}} = \mathsf{io}(c_{k_1}, \mathsf{w}_{j_{p'}}, \mathsf{w}_{|\sigma_P|+1}).\mathsf{io}(c_{k_2}, \mathsf{w}_{|\sigma_P|+1}, \mathsf{w}_{|\sigma_P|+2}) \ldots \mathsf{io}(c_{k_\ell}, \mathsf{w}_{|\sigma_P|+l-1}, \mathsf{w}_{|\sigma_P|+\ell})$$

and σ_P^* defined as expected relying on our semantics. Let $\mathsf{w}_* = \mathsf{w}$ and $\mathsf{w}'_* = \mathsf{w}_{|\sigma_P|+\ell}$.

Now, either there exists no frame σ_Q^* such that $(\mathsf{tr}^*, \sigma_Q^*) \in \mathsf{trace}(Q)$, in which case condition GUARDED$_P$ fails obviously, or such a frame exists. In this case, by construction of tr^*, we have that the test $\mathsf{w}_* = \mathsf{w}'_*$ is σ_P^*-valid, σ_P^*-guarded, and pulled-up in $(\mathsf{tr}^*, \sigma_P^*)$.

In order to conclude, it remains to show that $\mathsf{w}_*\sigma_Q^* \neq \mathsf{w}'_*\sigma_Q^*$. We already know that $\mathsf{w}\sigma_Q = \mathsf{w}_*\sigma_Q^*$. Suppose *ad absurdum* that $\mathsf{w}_*\sigma_Q^* = \mathsf{w}'_*\sigma_Q^*$. Because the sequences of channels that occur in $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w}')$ and $\mathsf{seq}_{\mathsf{tr}^*}(\mathsf{w}'_*)$ are the same, $\mathsf{w}'\sigma_Q$ and $\mathsf{w}'_*\sigma_Q^*$ are either constant and equal or of the form $\mathsf{f}(u, \mathsf{k}, r)$ with $\mathsf{f} \in \{\mathsf{senc}, \mathsf{aenc}, \mathsf{sign}\}$ and equal up to a renaming of their random seeds. In the first case, it is enough to conclude that $\mathsf{w}\sigma_Q = \mathsf{w}'\sigma_Q$, which is absurd. In the second case, $\mathsf{w}_*\sigma_Q^*$ and $\mathsf{w}'_*\sigma_Q^*$ being randomized, must have equal top-level random seeds, implying that this nonce was introduced before $\mathsf{io}(c_{i_{p'}}, \mathsf{w}_{j_{p'-1}}, \mathsf{w}_{j_{p'}})$ in the common prefix of their respective sequences. As the said prefix is also common to w and w' in tr, $\mathsf{w}\sigma_Q$ and $\mathsf{w}'\sigma_Q$ share the same top-level random seed and are thus equal, contradicting our hypothesis. Threfore: $\mathsf{w}_*\sigma_Q^* \neq \mathsf{w}'_*\sigma_Q^*$. Hence GUARDED_P is false.

Finally, if $\mathsf{w}\sigma_Q = \mathsf{w}'\sigma_Q$ but $\mathsf{w}\sigma_P \neq \mathsf{w}'\sigma_P$, conditions CONST_Q and GUARDED_Q will similarly fail. □

C.2. From trace equivalence to language equivalence

LEMMA C.4. *Let P and Q be two protocols in $\mathcal{C}_{\mathsf{pp}}$, the two real-time GPDA $\mathcal{A}^P_{\text{CONST}}$ and A^Q_{CONST} are such that:*

$$P \textit{ and } Q \textit{ satisfy conditions } \text{CONST}_P \textit{ and } \text{CONST}_Q \textit{ iff } \mathcal{L}(\mathcal{A}^P_{\text{CONST}}) = \mathcal{L}(\mathcal{A}^Q_{\text{CONST}}).$$

PROOF. We prove the two implications separately.

($\Rightarrow$) Assume that $\mathcal{L}(\mathcal{A}^P_{\text{CONST}}) \neq \mathcal{L}(\mathcal{A}^Q_{\text{CONST}})$, and consider w.l.o.g. a word $u \in \mathcal{L}(\mathcal{A}^P_{\text{CONST}}) \smallsetminus \mathcal{L}(\mathcal{A}^Q_{\text{CONST}})$. We distinguish two cases depending on whether u is accepted in state q_0 or q_{f}.

Case $u = c_{i_1}.c_{i_2}\dots c_{i_l}$ is accepted in q_0: In such a case, we built (tr, σ_P) as follows:

$$\mathsf{tr} = \mathsf{io}(c_{i_1}, \mathsf{start}, \mathsf{w}_1).\mathsf{io}(c_{i_2}, \mathsf{w}_1, \mathsf{w}_2)\dots\mathsf{io}(c_{i_l}, \mathsf{w}_{l-1}, \mathsf{w}_l)$$

with σ_P the substitution defined uniquely as expected from our semantics. We have that $(\mathsf{tr}, \sigma_P) \in \mathsf{trace}_{\mathsf{fwd}}(P)$ as the transition function δ fully captures input filtering and output of terms for protocols in $\mathcal{C}_{\mathsf{pp}}$. Since $u \notin \mathcal{L}(\mathcal{A}^Q_{\text{CONST}})$, we have that $(\mathsf{tr}, \sigma_Q) \notin \mathsf{trace}_{\mathsf{fwd}}(Q)$ for any substitution σ_Q, and thus the condition CONST_P fails.

Case u is accepted in q_{f}: In such a case, we also build a trace $(\mathsf{tr}, \sigma_P) \in \mathsf{trace}_{\mathsf{fwd}}(P)$ "corresponding" to u. The construction is a bit more involved. We have that u is of the form $c_{i_1}c_{i_2}\dots c_{i_k}c_{\mathsf{test}}c_{j_1}c_{j_2}\dots c_{j_l}c_{\mathsf{end}}$. Let $\mathsf{tr} = \mathsf{tr}_1.\mathsf{tr}_2$ with tr_1 and tr_2 defined as follows:

— $\mathsf{tr}_1 = \mathsf{io}(c_{i_1}, \mathsf{start}, \mathsf{w}_1).\mathsf{io}(c_{i_2}, \mathsf{w}_1, \mathsf{w}_2)\dots\mathsf{io}(c_{i_k}, \mathsf{w}_{k-1}, \mathsf{w}_k)$;
— $\mathsf{tr}_2 = \mathsf{io}(c_{j_1}, \mathsf{start}, \mathsf{w}_{k+1}).\mathsf{io}(c_{j_2}, \mathsf{w}_{k+1}, \mathsf{w}_{k+2})\dots\mathsf{io}(c_{j_l}, \mathsf{w}_{k+l-1}, \mathsf{w}_{k+l})$;

and σ_P is defined uniquely as expected from our semantics, as P is deterministic. We have that $(\mathsf{tr}, \sigma_P) \in \mathsf{trace}_{\mathsf{fwd}}(P)$ as the transition function δ fully captures input filtering and output of terms for protocols in $\mathcal{C}_{\mathsf{pp}}$. We can now define $\mathsf{w} = \mathsf{w}_k$ and $\mathsf{w}' = \mathsf{w}_{k+l}$. 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 $\mathsf{w}\sigma_P = \mathsf{w}'\sigma_P = \mathsf{c}$.

We know that $u = u_1c_{\mathsf{test}}u_2c_{\mathsf{end}} \notin \mathcal{L}(\mathcal{A}^Q_{\text{CONST}})$, and we may assume that u_1 and u_2 are both in $\mathcal{L}(\mathcal{A}^Q_{\text{CONST}})$. Indeed, otherwise, this means that there exists no substitution σ_Q such that $(\mathsf{tr}, \sigma_Q) \in \mathsf{trace}_{\mathsf{fwd}}(Q)$, and thus CONST_P fails, and the result holds. From now one, we assume that there exists σ_Q such that $(\mathsf{tr}, \sigma_Q) \in \mathsf{trace}_{\mathsf{fwd}}(Q)$.

Now, let $q \xrightarrow{c;\alpha/\beta} q'$ be the first failing transition in the run of u in $\mathcal{A}^Q_{\text{CONST}}$. We distinguish several cases:

(1) *Case* $q = q_0$ *and* $q' = q_{\mathsf{c}}$ *for some constant* c. In such a case $\mathsf{w}\sigma_Q \neq \mathsf{c}$ for any constant c, and $\mathsf{w}\sigma_Q$ is thus a guarded term. The condition CONST_P fails.
(2) *Case* $q = q_{\mathsf{c}}$ *and* $q' = q_{\mathsf{f}}$ *for some constant* c. In such a case $\mathsf{w}\sigma_Q = \mathsf{c}$ but $\mathsf{w}\sigma \neq \mathsf{c}$, making CONST_P fail once again.

Hence P and Q do not satisfy CONST_P. Symmetrically, if $u \in \mathcal{L}(\mathcal{A}^Q_{\text{CONST}}) \smallsetminus \mathcal{L}(\mathcal{A}^P_{\text{CONST}})$, the condition CONST_Q will fail.

($\Leftarrow$) If P and Q do not satisfy CONST_P (or CONST_Q), *i.e.* there exists a trace $(\mathsf{tr}, \sigma_P) \in \mathsf{trace}_{\mathsf{fwd}}(P)$ such that:

(1) either there exists no σ_Q such that $(\mathsf{tr}, \sigma_Q) \in \mathsf{trace}_{\mathsf{fwd}}(Q)$;
(2) or there exist $\mathsf{w}, \mathsf{w}' \in dom(\sigma_P)$ and a constant c such that $\mathsf{w}\sigma_P = \mathsf{w}'\sigma_P = \mathsf{c}$ but: either $\mathsf{w}\sigma_Q$ is not a constant, or $\mathsf{w}\sigma_Q$ is a constant but $\mathsf{w}\sigma_Q \neq \mathsf{w}'\sigma_Q$.

We consider such a trace of minimal length ℓ.

In the first case, thanks to minimality, we have that $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w}_\ell) = \mathsf{tr}$. From tr we build a word $u \in \mathcal{L}(\mathcal{A}^P_{\text{CONST}})$ by extracting the channels that occur in tr keeping the order. Since there does not exist σ_Q such that $(\mathsf{tr}, \sigma_Q) \in \mathsf{trace}_{\mathsf{fwd}}(Q)$, and the transition function δ of the automaton fully captures input filtering and output of terms for protocols in $\mathcal{C}_{\mathsf{pp}}$, we have that $u \notin \mathcal{L}(\mathcal{A}^Q_{\text{CONST}})$.

In the second case, thanks to minimality, we have that tr is actually made up of all the actions that occur in $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w})$ and $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w}')$ (note that these two sequences may share some actions). From tr, we built a word $u = u_1 c_{\mathsf{test}} u_2 c_{\mathsf{end}} \in \mathcal{L}(\mathcal{A}^P_{\text{CONST}})$ as follows:

— u_1 is obtained by extracting the channels that occur in $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w})$ preserving the order; and
— u_2 is obtained by extracting the channels that occur in $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w}')$ preserving the order;.

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 c_{test}, $\mathcal{A}^P_{\text{CONST}}$ is in q_0, the transition $q_0 \xrightarrow{c_{\mathsf{test}};\omega\mathsf{c}/\omega} q_{\mathsf{c}}$ is indeed possible as $\mathsf{w}\sigma_P = \mathsf{c}$; and similarly upon reading the c_{end}, $\mathcal{A}^P_{\text{CONST}}$ is in q_{c}, the transition $q_{\mathsf{c}} \xrightarrow{c_{\mathsf{end}};\omega\mathsf{c}/\omega} q_f$ is indeed possible as $\mathsf{w}'\sigma_P = \mathsf{c}$, hence $u \in \mathcal{L}(\mathcal{A}^P_{\text{CONST}})$. What remains to show is that $u \notin \mathcal{L}(\mathcal{A}^Q_{\text{CONST}})$. We distinguish two cases:

— *Case* $\mathsf{w}\sigma_Q$ *is not a constant.* In such a case, no transition $q_0 \xrightarrow{c_{\mathsf{test}};\omega\mathsf{c}/\omega} q_{\mathsf{c}}$ will be possible after u_1
— *Case* $\mathsf{w}\sigma_Q$ *is a constant* c *but* $\mathsf{w}\sigma_Q \neq \mathsf{w}'\sigma_Q$. In such a case, the transition $q_0 \xrightarrow{c_{\mathsf{end}};\omega\mathsf{c}/\omega} q_c$ will not be possible after u_2.

Hence u cannot belong to $\mathcal{L}(\mathcal{A}^Q_{\mathsf{const}})$. This allows us to conclude. □

LEMMA C.5. *Let P and Q be two protocols in $\mathcal{C}_{\mathsf{pp}}$, the two real-time GPDA $\mathcal{A}^P_{\text{GUARDED}}$ and A^Q_{GUARDED} are such that:*

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

PROOF. We prove the two directions separately.

($\Rightarrow$) Assume that $\mathcal{L}(\mathcal{A}^P_{\text{GUARDED}}) \neq \mathcal{L}(\mathcal{A}^Q_{\text{GUARDED}})$, and consider w.l.o.g. a word $u \in \mathcal{L}(\mathcal{A}^P_{\text{GUARDED}}) \smallsetminus \mathcal{L}(\mathcal{A}^Q_{\text{GUARDED}})$. We distinguish two cases depending on whether the word u is accepted in state q_0 or q_{f}.

Case $u = c_{i_1} c_{i_2} \dots c_{i_l}$ *is accepted in* q_0*:* In such a case, we built (tr, σ_P) as follows:

$$\mathsf{tr} = \mathsf{io}(c_{i_1}, \mathsf{start}, \mathsf{w}_1).\mathsf{io}(c_{i_2}, \mathsf{w}_1, \mathsf{w}_2) \dots \mathsf{io}(c_{i_l}, \mathsf{w}_{l-1}, \mathsf{w}_l)$$

with σ_P the substitution defined uniquely as expected from our semantics. We have that $(\mathsf{tr}, \sigma_P) \in \mathsf{trace}_{\mathsf{fwd}}(P)$ as the transition function δ fully captures input filtering and output of terms for protocols in $\mathcal{C}_{\mathsf{pp}}$. Since $u \notin \mathcal{L}(\mathcal{A}^Q_{\text{CONST}})$, we have that $(\mathsf{tr}, \sigma_Q) \notin \mathsf{trace}_{\mathsf{fwd}}(Q)$ for any substitution σ_Q, and thus the condition GUARDED_P fails.

Case u *is accepted in* q_{f}*:* In such a case, we also build a trace $(\mathsf{tr}, \sigma_P) \in \mathsf{trace}_{\mathsf{fwd}}(P)$ "corresponding" to u. The construction is a bit more involved. We have that u is of the form: $c_{i_1} c_{i_2} \dots c_{i_k} c^{i_0}_{\mathsf{fork}} c_{j_1} c_{j_2} \dots c_{j_l} c_{\mathsf{test}} c_{p_1} c_{p_2} \dots c_{p_m} c_{\mathsf{end}}$. Let $\mathsf{tr} = \mathsf{tr}_0.\mathsf{tr}_1.\mathsf{tr}_2$ with tr_0, tr_1 and tr_2 defined as follows:

— $\mathsf{tr}_0 = \mathsf{io}(c_{i_1}, \mathsf{start}, \mathsf{w}_1).\mathsf{io}(c_{i_2}, \mathsf{w}_1, \mathsf{w}_2) \dots \mathsf{io}(c_{i_k}, \mathsf{w}_{k-1}, \mathsf{w}_k).\mathsf{io}(c_{i_0}, \mathsf{w}_k, \mathsf{w}_{k+1})$;
— $\mathsf{tr}_1 = \mathsf{io}(c_{j_1}, \mathsf{w}_{k+1}, \mathsf{w}_{k+2}).\mathsf{io}(c_{j_2}, \mathsf{w}_{k+2}, \mathsf{w}_{k+3}) \dots \mathsf{io}(c_{j_l}, \mathsf{w}_{k+l}, \mathsf{w}_{k+l+1})$;
— $\mathsf{tr}_2 = \mathsf{io}(c_{p_1}, \mathsf{w}_{k+1}, \mathsf{w}_{k+l+2}).\mathsf{io}(c_{p_2}, \mathsf{w}_{k+l+2}, \mathsf{w}_{k+l+3}) \dots \mathsf{io}(c_{p_m}, \mathsf{w}_{k+l+m}, \mathsf{w}_{k+l+m+1})$.

and σ_P is uniquely defined as expected from our semantics, as P is determinitic. We have that $(\mathsf{tr}, \sigma_P) \in \mathsf{trace}_{\mathsf{fwd}}(P)$ as the transition function fully captures input filtering and output of terms for protocols in $\mathcal{C}_{\mathsf{pp}}$. We can now define $\mathsf{w} = \mathsf{w}_{k+l+1}$ and $\mathsf{w}' = \mathsf{w}_{k+l+m+1}$. The test is σ_P-guarded (the indice k associated to the stack symbol (fork, k) is indeed stricly positive), σ_P-valid (since the last transition from q_2 to q_{f} requires the stack to be identical to the stack before reading c_{test}), and pulled-up in (tr, σ_P) (since the fork tiles allow us to control the first time the top-level random seed of $\mathsf{w}\sigma_P$ appears in the frame).

We know that $u = u_0 c^{i_0}_{\mathsf{fork}} u_1 c_{\mathsf{test}} u_2 c_{\mathsf{end}} \notin \mathcal{L}(\mathcal{A}^Q_{\text{GUARDED}})$, and we may assume w.l.o.g. that $u_0 c_{i_0} u_1$ and $u_0 c_{i_0} u_2$ are both in $\mathcal{L}(\mathcal{A}^Q_{\text{GUARDED}})$. Indeed, otherwise this means that there exists no frame σ_Q such that $(\mathsf{tr}, \sigma_Q) \in \mathsf{trace}_{\mathsf{fwd}}(Q)$, and thus GUARDED_P fails, and the result holds. From now on, we assume that there exists σ_Q such that $(\mathsf{tr}, \sigma_Q) \in \mathsf{trace}_{\mathsf{fwd}}(Q)$.

Now, let $q \xrightarrow{c;\alpha/\beta} q'$ be the first failing transition in the run of u in $\mathcal{A}^Q_{\text{GUARDED}}$. We distinguish several cases:

(1) *Case* $q = q_0$ *and* $q' = q_1$. Since we have already assume that $u_0 c_{i_0} u_1$ is in $\mathcal{L}(\mathcal{A}^Q_{\text{GUARDED}})$, this means that the required transition does not exist because $||v^j_i|| = 0$. In such a case, the test $\mathsf{w} = \mathsf{w}'$ (even if it was σ_Q-valid and σ_Q-guarded) can not be a pulled-up one in (tr, σ_Q). Thus the condition GUARDED_P fails.
(2) *Case* $q = q_1$ *and* $q' = q_1$. In such a case, this means that a fork tile cannot be unstacked, meaning that the corresponding test (even if it was σ_Q-valid and σ_Q-guarded) will not be pulled-up in (tr, σ_Q), and GUARDED_P is false.
(3) *Case* $q = q_1$ *and* $q' = q_2$. In such a case, the problem occurs due to the fact that the fork tile is not at the top of the stack upon becoming test. The corresponding test $\mathsf{w} = \mathsf{w}'$ will not be σ_Q-valid since $\mathsf{w}\sigma_Q$ will contain a random seed that has been generated after the "forking point", and thus this random seed can not occur in $\mathsf{w}'\sigma_Q$. Thus, the condition GUARDED_P fails.
(4) *Case* $q = q_2$ *and* $q' = q_{\mathsf{f}}$. In such a case, the test tile is not at the top of the stack upon reading the last letter of the word. The test is not σ_Q-valid. The stack at this point, without the test tile, is not identical to the stack before the fork tile turning test, making GUARDED_Q fail.

Hence GUARDED_Q fails as soon as $u \notin \mathcal{L}(\mathcal{A}^Q_{\text{GUARDED}})$.

($\Leftarrow$) If P and Q do not satisfy conditions GUARDED_P (or GUARDED_Q), *i.e.* there exists a trace $(\mathsf{tr}, \sigma_P) \in \mathsf{trace}_{\mathsf{fwd}}(P)$ such that:

(1) either there exists no σ_Q such that $(\mathsf{tr}, \sigma_Q) \in \mathsf{trace}_{\mathsf{fwd}}(Q)$;
(2) or (such a σ_Q exists) and there exists $\mathsf{w}, \mathsf{w}' \in dom(\sigma_P)$ such that the test $\mathsf{w} = \mathsf{w}'$ is σ_P-guarded and σ_P-valid, and pulled-up in (tr, σ_P), and
   — either $\mathsf{w} = \mathsf{w}'$ is not σ_Q-valid,
   — or $\mathsf{w} = \mathsf{w}'$ is not σ_Q-guarded,
   — or $\mathsf{w} = \mathsf{w}'$ is not pulled-up in (tr, σ_Q).

We consider such a trace of minimal length ℓ.

In the first case, thanks to the minimality, we have that $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w}_\ell) = \mathsf{tr}$. From tr we build a word $u \in \mathcal{L}(\mathcal{A}^P_{\text{GUARDED}})$ by extracting the channels that occur in tr keeping the order. Since there does not exist σ_Q such that $(\mathsf{tr}, \sigma_Q) \in \mathsf{trace}_{\mathsf{fwd}}(Q)$, and the transition function δ of the automaton fully captures input filtering and output of terms for protocols in $\mathcal{C}_{\mathsf{pp}}$, we have that $u \notin \mathcal{L}(\mathcal{A}^Q_{\text{GUARDED}})$.

In the second case, thanks to minimality, we have that tr is actualy made up of all the actions that occur in $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w})$ and $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w}')$. These two sequences have a maximal common prefix pref that is not empty. Actually, we have that:

$$\mathsf{pref} = \mathsf{io}(c_{i_1}, \mathsf{start}, \mathsf{w}_1).\mathsf{io}(c_{i_2}, \mathsf{w}_1, \mathsf{w}_2) \dots \mathsf{io}(c_{i_p}, \mathsf{w}_{p-1}, \mathsf{w}_p) \text{ for some } p \geq 1.$$

From tr, we build a word $u = c_{i_1} c_{i_2} \dots c_{i_{p-1}} c^{i_p}_{\mathsf{fork}} u_1 c_{\mathsf{test}} u_2 c_{\mathsf{end}} \in \mathcal{L}(\mathcal{A}^P_{\text{GUARDED}})$ as follows:

— u_1 is obtained by extracting the channels that occur in $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w})$ after the prefix $c_{i_1} c_{i_2} \dots c_{i_{p-1}} c_{i_p}$; and
— u_2 is obtained by extracting the channels that occur in $\mathsf{seq}_{\mathsf{tr}}(\mathsf{w}')$ after the prefix $c_{i_1} c_{i_2} \dots c_{i_{p-1}} c_{i_p}$.

As the transition function δ fully captures input filtering and output of terms for protocols in $\mathcal{C}_{\mathsf{pp}}$, and since $\mathsf{w} = \mathsf{w}'$ is a test that is σ_P-guarded, σ_P-valid and pulled-up in (tr, σ_P), we get that $u \in \mathcal{L}(\mathcal{A}^P_{\text{GUARDED}})$. What remains to show is that $u \notin \mathcal{L}(\mathcal{A}^Q_{\text{GUARDED}})$. We distinguish two cases:

— *Case* $\mathsf{w} = \mathsf{w}'$ *is not* σ_Q*-valid.* In such a case, even if after reading the first part of u, *i.e.* $c_{i_1} c_{i_2} \dots c_{i_{p-1}} c^{i_p}_{\mathsf{fork}} u_1 c_{\mathsf{test}}$, we reach q_2, then we will fail to read the remaining of the word to end in q_{f}.
— *Case* $\mathsf{w} = \mathsf{w}'$ *is* σ_Q*-valid but* $\mathsf{w} = \mathsf{w}'$ *is not* σ_Q*-guarded.* In such a case, this means that $\mathsf{w}\sigma_Q$ is a constant, and the run will stop in q_0 after reading $c_{i_1} c_{i_2} \dots c_{i_{p-1}}$. This comes from the fact that it is not possible to go from q_0 to q_1 adding a tile (fork^j_i, k) with $k = 0$.
— *Case* $\mathsf{w} = \mathsf{w}'$ *is* σ_Q*-valid,* σ_Q*-guarded, but not pulled-up in* (tr, σ_Q). The fact that the test is σ_Q-valid but not pulled-up means that the run will stop in q_1 after reading because of the presence of a tile (fork^j_i, k) in the stack that can not go down anymore.

Hence u cannot belong to $\mathcal{L}(\mathcal{A}^Q_{\text{GUARDED}})$. This allows us to conclude. □