Trace equivalence of protocols for an unbounded number of sessions

Rémy Chrétien

December 2012

Research report LSV-12-22

Laboratoire Spécification & Vérification

École Normale Supérieure de Cachan
61, avenue du Président Wilson
94235 Cachan Cedex France

Trace equivalence of protocols for an unbounded number of sessions

Rémy Chrétien, under the supervision of Véronique Cortier and Stéphanie Delaune, SECSI/LSV and LORIA

19th August 2012

This report is meant to serve as basis for a later research article and is for this reason written in English.

The general context

Secure design of communication protocols in order to ensure the authentication of electronic agents or the safety of secret data is known to be difficult and fairly error-prone. Symbolic frameworks such as the Dolev-Yao model [17] and later various process algebra [1, 2] have proven themselves valuable for finding attacks and assessing the security of these protocols. Several tools have thus been developed to answer the need of automated verification: ProVerif [8], AVISPA [6] and Scyther [14] rely on various formal methods to prove that a range of security properties holds in protocols. Most verified properties, such as secrecy or authentication, actually relate to reachability problems, *i.e.* deciding whether an attacker is able learn a particular information, such as a secret cryptographic key.

The research problem

We aim at providing a decidability result for the problem of trace equivalence between protocols with one variable for an unbounded number of sessions. Being able to decide the equivalence of protocols naturally arises when dealing with practical applications such as electronic voting [15] or the electronic passport [4] which require a notion of privacy to be expressed.

The problem of deciding reachability has been thoroughly studied for an unbounded number of sessions and proven to be undecidable under various restraints on nonces [19, 12, 18, 3]. Nevertheless some fragments were shown to be decidable, either by tagging [9, 20] or by restricting the number of blind-copies [13]. On the other hand, trace equivalence itself is only proven to be decidable for a bounded number of sessions [11], among other restrictions.

The objective of this report is to provide the first results of decidability for certain classes of protocols, by lifting the approach followed by Comon-Lundh and Cortier [13] to trace equivalence.

Our contribution

Because deciding trace equivalence appears to be more complex than deciding the reachability of a term, as shown in Section 3.3, we used the restrictions provided in Comon-Lundh and Cortier [13] as a starting point and translated it back to a process algebra inspired from variants of the applied pi-calculus (see *e.g.* [16, 7]).

This initial set of constraints lead us to a first result of undecidability of trace equivalence under scare restrictions: one variable and symmetric encryption are indeed enough.

Consequently, we restrained our class of protocols a step further by making the protocols *deterministic* in some sense and preventing it from disclosing secret keys. This tighter class of protocols was then shown to be decidable after reduction to an equivalence between deterministic pushdown automata [21].

Arguments supporting its validity

Our first class of protocols is proven to be undecidable with regards to trace equivalence and appears to fit in the intended class described in [13]. This makes our class of protocol an explicit witness of the increased complexity of the decision problem for trace equivalence compared to reachability.

Its decidable subclass gives on the other hand an idea of the gap between a undecidable class and a decidable fragment. In particular it enlightens potential causes of undecidability: *non-determinism* of the protocol and disclosure of keys during an execution. Moreover, this result lays a bridge between the quite recent concern of protocol equivalence and the older issue of equivalence of context-free grammars in language theory. This decidable class in nonetheless not expected to serve practical means yet: even though the tighter restrictions can seemingly be weakened, the final result relies on a procedure for which there exists no interesting complexity bound [22] nor implementation.

Summary and future work

The immediate next trail is to extend the decidable class we exposed to drop the tightest restrictions and enrich its set of primitives to obtain a fragment expressive enough to offer a practical use in protocol verification. Looking at a broader horizon, an implementation of our decision would be interesting. In parallel, applying the same approach to tagged protocols could lead to new and more reasonable classes of protocols.

1 Introduction

Secure design of communication protocols in order to ensure the authentication of electronic agents or the safety of secret data is known to be difficult and fairly error-prone. To alleviate the weight of this task, symbolic frameworks such as the Dolev-Yao model [17] and later various process algebra [1, 2] have proven themselves valuable for finding attacks and assessing the security of these protocols. For example, a flaw has been discovered in the Single-Sign-On protocol used *e.g.* by Google Apps [5]. It has been shown that a malicious application could very easily gain access to any other application (*e.g.* Gmail or Google Calendar) of their users. This flaw has been found when analysing the protocol using formal methods, abstracting messages by a term algebra and using the AVISPA platform [6]. Several tools have thus been developed to answer the need of automated verification: ProVerif [8], AVISPA and Scyther [14] rely on various formal methods to prove that a range of security properties holds in protocols. Most verified properties, such as secrecy or authentication, actually relate to reachability problems, *i.e.* deciding whether an attacker is able learn a particular information, such as a secret cryptographic key.

Nevertheless, practical applications for which privacy is a crucial feature, such as electronic voting [15] or the electronic passport [4], actually require different and specific properties to be checked. For this reason, the need for studying logical concepts able to express privacy-related properties arose. Those properties can indeed be expressed by the notion of equivalence of protocols, *i.e.* deciding whether an attacker can distinguish in some way between two protocols. In particular, trace equivalence focuses on the (un)ability for an attacker to find seemingly identical execution traces for two different protocols and provides a well-suited conceptual framework to address privacy issues.

Related work. Reachability itself has been shown to be undecidable for an unbounded number of sessions, with [19, 12] or without [18, 3] severe constraints on the use of nonces. Conversely, several classes of protocols whose reachability is decidable were exposed, relying for instance on tagging [9, 20] or restraints on blind-copies of messages [13] to enforce termination of the different resolution procedures used to solve it. Trace equivalence, on the other hand, has only be proven to be decidable under the restriction, notably, of a bounded number of sessions [11].

Our contributions. In this report, we investigate the decidability of the trace equivalence for two classes of protocols for an unbounded number of sessions. We use a variant of the applied-pi calculus as our basic modelling formalism [1] which has the advantage of being based on well-understood concepts.

In particular, in Section 4.2, we prove that a class of protocol analogous to the logical fragment discussed in Comon-Lundh and Cortier [13], although decidable with regards to reachability, is undecidable for trace equivalence.

A more constrained subclass is then studied in Section 4.3. Its decidability relies on the equivalence of deterministic pushdown automata, proven to be decidable by Sénizergues [21]. This class is nonetheless expressive enough to include non-trivial protocols (see Example 7 whose witness of non-equivalence informally requires to compute the least common factor of two integers).

2 Messages and attacker capabilities

2.1 Messages

For modelling messages, we consider an arbitrary term algebra, which provides a lot of flexibility in terms of which cryptographic primitives can be modelled. More precisely, we consider a *signature* Σ made of a set of *function symbols* Σ together with arities of the form $ar(\mathsf{f}) = k$ where $\mathsf{f} \in \Sigma$, and $k \in \mathbb{N}$. We consider an infinite set of *variables* $\mathcal{X}$ and an infinite set of *names* $\mathcal{N}$ which are used for representing keys, nonces, ...

Terms are defined as names, variables, and function symbols applied to other terms. For $\mathcal{A} \subseteq \mathcal{X} \cup \mathcal{N}$, the set of terms built from $\mathcal{A}$ by applying function symbols in Σ is denoted by $\mathcal{T}(\Sigma, \mathcal{A})$. A term u is said to be a *ground* term if it does not contain any variable.

For our cryptographic purposes, it is useful to distinguish a subset Σ_{pub} of Σ, made of *public symbols*, *i.e.* the symbols made available to the attacker. A *recipe* is a term in $\mathcal{T}(\Sigma_{\mathsf{pub}}, \mathcal{X} \cup \mathcal{N})$, that is, a term containing no private (non-public) symbols. Moreover, to model algebraic properties of cryptographic primitives, we define an *equational theory* by a finite set E of equations $u = v$ with $u, v \in \mathcal{T}(\Sigma, \mathcal{X})$ (note that u, v do not contain names). We define $=_{\mathsf{E}}$ to be the smallest equivalence relation on terms, that contains E and that is closed under application of function symbols and substitutions of terms for variables. We restrict ourselves to sets of equations which can be oriented from left to right as a convergent rewriting system: given a term u, we denote by $u\downarrow$ its normal form.

In particular we are interested in the following class S of signature Σ defined as

$$\begin{aligned}\Sigma &= \Sigma_{\mathsf{pub}} \uplus \Sigma_{\mathsf{prv}}\\ \Sigma_{\mathsf{pub}} &= \{\mathsf{senc}/2, \mathsf{sdec}/2\} \cup H_{\mathsf{pub}} \cup A_{\mathsf{pub}}\\ \Sigma_{\mathsf{prv}} &= H_{\mathsf{prv}} \cup A_{\mathsf{prv}}\end{aligned}$$

with the equational theory E:

$$E = \{\mathsf{sdec}(\mathsf{senc}(x, y), y) = x\}$$

where H_{pub} and H_{prv} (resp. A_{pub} and A_{prv}) are set of public and private unary function symbols (resp. public and private constants), whereas senc and sdec represent symmetric encryption and decryption.

Substitutions are written $\sigma = \{x_1 \triangleright u_1, \ldots, x_n \triangleright u_n\}$ with $dom(\sigma) = \{x_1, \ldots, x_n\}$, and $img(\sigma) = \{u_1, \ldots, u_n\}$. The size of σ is defined as the cardinal of its domain: $|\sigma| = |dom(\sigma)|$. The application of a substitution σ to a term u is written $u\sigma$. A *most general unifier* of two terms u and v is a substitution denoted by $mgu(u, v)$. We write $mgu(u, v) = \bot$ when u and v are not unifiable.

2.2 Attacker capabilities

To represent the knowledge of an attacker (who may have observed a sequence of messages $M_1, \ldots, M_\ell$), we use the concept of *frame*. A frame $\phi = \mathsf{new}\ \tilde{n}.\sigma$ consists of a finite set $\tilde{n} \subseteq \mathcal{N}$ of *restricted* names (those unknown to the attacker), and a substitution σ of the form $\{y_1 \triangleright M_1, \ldots, y_\ell \triangleright M_\ell\}$ where each M_i is a ground term. The variables y_i enable an attacker to refer to each M_i. The *domain* of the frame ϕ, written $dom(\phi)$, is $dom(\sigma) = \{y_1, \ldots, y_\ell\}$.

In the frame $\phi = \mathsf{new}\ \tilde{n}.\sigma$, the names $\tilde{n}$ are bound in σ and can be renamed. Moreover names that do not appear in ϕ can be added or removed from $\tilde{n}$. In particular, we can always assume that two frames share the same set of restricted names. Thus, in the definition below, we will assume w.l.o.g. that the two frames ϕ_1 and ϕ_2 have the same set of restricted names.

Definition 1 (static equivalence). *We say that two frames $\phi_1 = \mathsf{new}\ \tilde{n}.\sigma_1$ and $\phi_2 = \mathsf{new}\ \tilde{n}.\sigma_2$ are* statically equivalent*, $\phi_1 \sim_{\mathsf{E}} \phi_2$, when $dom(\phi_1) = dom(\phi_2)$, and for all recipes M, N such that $fn(M, N) \cap \tilde{n} = \emptyset$, we have that: $M\sigma_1 =_{\mathsf{E}} N\sigma_1$ if, and only if, $M\sigma_2 =_{\mathsf{E}} N\sigma_2$.*

Intuitively, two frames are equivalent when the attacker cannot see the difference between the two situations they represent, *i.e.*, his ability to distinguish whether two recipes M, N produce the same term does not depend on the frame.

Example 1. Consider the frames

$$\phi_1 = \mathsf{new}\ \tilde{n}.\sigma_1;\ \phi_2 = \mathsf{new}\ \tilde{n}.\sigma_2;\ \phi_3 = \mathsf{new}\ \tilde{n}.\sigma_3;\ \phi_4 = \mathsf{new}\ \tilde{n}.\sigma_4$$

where a is a public constant of the signature Σ and

$$\begin{cases} \tilde{n} = \{k, k', n\} & \sigma_1 = \{x_1 \triangleright k, x_2 \triangleright \mathsf{senc}(a, k)\} \\ \sigma_2 = \{x_1 \triangleright k, x_2 \triangleright a\} & \sigma_3 = \{x_1 \triangleright k, x_2 \triangleright n\} \\ \sigma_4 = \{x_1 \triangleright k, x_2 \triangleright \mathsf{senc}(a, k')\} & \end{cases}$$

Then, with our equational theory E:

– $\phi_1 \not\sim_{\mathsf{E}} \phi_2$ as we consider the recipes $M = \mathsf{senc}(a, x_1)$ and $N = x_2$. Indeed, $fn(M, N) \cap \tilde{n} = \emptyset$, $M\sigma_1 =_{\mathsf{E}} N\sigma_1$ but $M\sigma_2 \neq_{\mathsf{E}} N\sigma_2$.
– $\phi_3 \sim_{\mathsf{E}} \phi_4$ as without key k', the attacker is unable to observe the differences between the content of x_2 and a nonce. Note that proofs of static equivalence are in general difficult to produce because of the quantification on all the possible recipes the attacker can use.

3 Models for protocols

In this section, we introduce the cryptographic process calculus that we will use for describing protocols. Several well-studied calculi already exist to analyse security protocols and privacy-type properties (*e.g.* [2, 1]). Our calculus is actually inspired from other calculi (*e.g.* [16, 7]) adapted to allow syntactic filtering.

3.1 Syntax

The intended behaviour of a protocol can be modelled by a *process* defined by the grammar given below (u is a term that may contain variables, n and c are names).

$P, Q :=$	0	null process
	$\mathsf{in}(c, u).P$	reception on channel c
	$\mathsf{out}(c, u).P$	emission on channel c
	$P \mid Q$	parallel composition
	$!P$	replication
	$\mathsf{new}\ n.P$	fresh name generation

The process "$\mathsf{in}(c, u).P$" expects a message m of the form u on channel c and then behaves like $P\sigma$ where σ is such that $m = u\sigma$. The process "$\mathsf{out}(c, u).P$" emits u on channel c, and then behaves like P. The variables that occur in u will be instantiated when the evaluation will take place. The other operators are standard.

Even though our syntax includes fresh name generation, we will not use it in any of the later classes of protocols and focus on processes without names, but only constants.

Sometimes, for the sake of clarity, we will omit the null process. We write $fv(P)$ for the set of *free variables* that occur in P, *i.e.* the set of variables that are not in the scope of an input. We consider *ground processes*, *i.e.* processes P such that $fv(P) = \emptyset$. A *protocol* is a couple (P, ϕ_0) where P is a ground process and ϕ_0 the *initial frame*, *i.e.* a frame gathering the initial knowledge of the attacker.

Example 2. Consider the following basic example of communication between an electronic passport (agent A) and a server (agent S). Here is the protocol described in semi-informal Alice & Bob notation:

$$\begin{array}{ll} A \to S : \{id_A\}_{k_{AS}} & (a) \\ S \to A : \{x\}_{k_{AS}} \to h(x), \{x\}_{k_{\mathsf{auth}}} & (b) \end{array}$$

The passport, A, identifies itself to the server S by sending its identifier id_A encrypted by a symmetric secret key k_{AS}, shared between S and A. The terminal answers to any request whose key is indeed k_{AS} by issuing a receipt to the passport and producing a certificate consisting in the identity encrypted with k_{auth}. The representation of this protocol in our syntax would be:

$$\begin{array}{rll} P_A = & \mathsf{in}(c_A^A, \mathsf{start}).\mathsf{out}(c_A^A, \mathsf{senc}(id_A, k_{AS})) & (1) \\ & | \,!\, \mathsf{in}(c_A^S, \mathsf{senc}(x, k_{AS})).\mathsf{out}(c_A^S, h(x)) & (2) \\ & | \,!\, \mathsf{in}(c_S, \mathsf{senc}(x, k_{AS})).\mathsf{out}(c_S, \mathsf{senc}(x, k_{\mathsf{auth}})) & (3) \end{array}$$

where start, id_A and h are public constants and function symbol in the signature. Branch (1) directly mimics the communication (a), except a starting input with a public constant start is asked to actually initiate a protocol session. As our syntax does not allow pairing, the communication (b) is then replicated with the two parallel branches (2) and (3). Then $(P_A, \emptyset)$ is a protocol.

3.2 Configuration and execution model

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

- $\mathcal{E}$ is a finite set of names that represents the names restricted in $\mathcal{P}$ and σ;
- $\mathcal{P}$ is a multiset of processes We write $P \cup \mathcal{P}$ instead of $\{P\} \cup \mathcal{P}$.
- $\sigma = \{x_1 \triangleright u_1, \ldots, x_n \triangleright u_n\}$ where $u_1, \ldots, u_n$ are ground terms (the messages observed by the attacker), and $x_1, \ldots, x_n$ are variables.

The *initial configuration* of a protocol (P, ϕ_0) is the configuration $(\mathcal{E}_0; \{P\}; \sigma_0)$ if $\phi_0 = \mathsf{new}\ \mathcal{E}_0.\sigma_0$.

The communication system is formally defined by the rules of Figure 1. The In rule allows the attacker to make a process progress by feeding it a term he built with publicly available terms and symbols. The Out rule let the attacker gain knowledge as soon as a message is sent by a process by adding it to the substitution of its current configuration. The other rules are quite standard.

Let $\mathcal{A}$ be the alphabet of actions where the special symbol $\tau \in \mathcal{A}$ represents an unobservable action. For every $\ell \in \mathcal{A}$, the relation $\xrightarrow{\ell}$ has been

$$
\begin{array}{lrcl}
\textsc{In} & (\mathcal{E}; \mathsf{in}(c,u).P \cup \mathcal{P}; \sigma) & \xrightarrow{\mathsf{in}(c,R)} & (\mathcal{E}; P\theta \cup \mathcal{P}; \sigma) \\
 & & & \text{where } \begin{cases} R \text{ is a recipe} \\ \theta = mgu(u, R\sigma\downarrow) \neq \bot \end{cases} \\
\textsc{Out} & (\mathcal{E}; \mathsf{out}(c,u).P \cup \mathcal{P}; \sigma) & \xrightarrow{\mathsf{out}(c, x_{|\sigma|+1})} & (\mathcal{E}; P \cup \mathcal{P}; \sigma \cup \{x_{|\sigma|+1} \triangleright u\}) \\
\textsc{Par} & (\mathcal{E}; P_1 \mid P_2 \cup \mathcal{P}; \sigma) & \xrightarrow{\tau} & (\mathcal{E}; P_1 \cup P_2 \cup \mathcal{P}; \sigma) \\
\textsc{Repl} & (\mathcal{E}; !P \cup \mathcal{P}; \sigma) & \xrightarrow{\tau} & (\mathcal{E}; P \cup !P \cup \mathcal{P}; \sigma) \\
\textsc{New} & (\mathcal{E}; \mathsf{new}\ n.P \cup \mathcal{P}; \sigma) & \xrightarrow{\tau} & (\mathcal{E} \cup \{n'\}; P\{^{n'}/_{n}\} \cup \mathcal{P}; \sigma) \\
 & & & \text{where } n' \text{ is a fresh name}
\end{array}
$$

Fig. 1. Transition system.

defined in Figure 1. For every $w \in \mathcal{A}^*$ the relation $\xrightarrow{w}$ on configurations is defined in the usual way. By convention $K \xrightarrow{\epsilon} K$ where ϵ denotes the empty word. For every $s \in (\mathcal{A} \setminus \{\tau\})^*$, the relation $\stackrel{s}{\Longrightarrow}$ on configurations is defined by: $K \stackrel{s}{\Longrightarrow} K'$ if, and only if, there exists $w \in \mathcal{A}^*$ such that $K \xrightarrow{w} K'$ and s is obtained from w by erasing all occurrences of τ. Intuitively, $K \stackrel{s}{\Longrightarrow} K'$ means that K transforms into K' by experiment s.

Example 3. Consider the following execution of the protocol $(P_A, \emptyset)$ described in Example 2:

$$
\begin{array}{l}
\mathsf{tr} = K_0 \xrightarrow{\tau}\xrightarrow{\tau}\xrightarrow{\tau} \xrightarrow{\mathsf{in}(c_A^A, \mathsf{start})} (\emptyset; \mathcal{P}_1; \emptyset) \xrightarrow{\mathsf{out}(c_A^A, x_1)} (\emptyset; \mathcal{P}_2; \sigma_2) \xrightarrow{\tau} \xrightarrow{\mathsf{in}(c_A^S, x_1)} (\emptyset; \mathcal{P}_3; \sigma_2) \\
\xrightarrow{\mathsf{out}(c_A^S, x_2)} (\emptyset; \mathcal{P}_4; \sigma_4) \xrightarrow{\tau} \xrightarrow{\mathsf{in}(c_S, x_1)} (\emptyset; \mathcal{P}_5; \sigma_4) \xrightarrow{\mathsf{out}(c_S, x_3)} (\emptyset; \mathcal{P}_6; \sigma)
\end{array}
$$

where:

$$
\begin{cases}
K_0 = (\emptyset; \{P_A\}; \emptyset), \text{ the initial configuration of } (P_A, \emptyset) \\
\sigma_2 = \{x_1 \triangleright \mathsf{senc}(id_A, k_{AS})\} \\
\sigma_4 = \{x_1 \triangleright \mathsf{senc}(id_A, k_{AS}), x_2 \triangleright h(id_A)\} \\
\sigma = \{x_1 \triangleright \mathsf{senc}(id_A, k_{AS}), x_2 \triangleright h(id_A), x_3 \triangleright \mathsf{senc}(id_A, k_{\mathsf{auth}})\}
\end{cases}
$$

Three uses of rule Par are first needed to separate P_A into three distinct processes. The branch (1) in P_A first evolves by receiving the message start on channel c_A^A (rule In). It then outputs a new term by rule Out, $\mathsf{senc}(id_A, k_{AS})$, stored in the substitution σ_2. Note that, as P_A does not contain any new instruction, there are no restricted names and the frame is just the associated substitution. The rule Repl enables us to make a new instance of (2) without the bang (!). It can now receive an input by In: x_1, which, in the current frame, corresponds to $\mathsf{senc}(id_A, k_{AS})$ and consequently outputs $h(id_A)$ on channel c_A^S, stored as x_2, thanks to the Out rule. The same applies to branch (3) which after an application of Repl receives x_1 on channel c_S and emits $\mathsf{senc}(id_A, k_{\mathsf{auth}})$, stored in the frame σ as x_3.

3.3 Reachability

Most common notions of security, such as secrecy or authentication rely on the ability for an attacker to learn a particular information. Formally these properties can be modelled with the notion of reachability, *i.e.* whether a term in our grammar can be reached through some execution of a protocol.

For every configuration K, we define its set of traces, each trace consisting in a sequence of actions together with the sequence of sent messages:

$$\mathsf{trace}(K) = \{(s, \mathsf{new}\,\mathcal{E}.\sigma) \mid K \xRightarrow{s} (\mathcal{E}; \mathcal{P}; \sigma) \text{ for some configuration } (\mathcal{E}; \mathcal{P}; \sigma)\}.$$

Definition 2 (reachability). *A term u is said to be reachable in a protocol (P, ϕ_0) if there exists $(s, \mathsf{new}\ \mathcal{E}.\sigma) \in \mathsf{trace}(K_0)$ and $i \in \{1, \ldots, |\sigma|\}$ such that $x_i\sigma = u$; where K_0 is the initial configuration of the protocol $(P|!\mathsf{in}(c, x).\mathsf{out}(c, x), \phi_0)$ and c is channel not used in P.*

Informally, a term is reachable if the attacker is able to build it from the knowledge gained from some execution of the protocol.

3.4 Trace equivalence

Privacy-type security properties such as those studied in this paper are often formalised using behavioural equivalence (see *e.g.* [15, 4, 10]). In this paper, we consider the notion of trace equivalence.

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

Definition 3 (trace equivalence). *Let K_A and K_B be two configurations, $K_A \sqsubseteq K_B$ if for every $(\mathsf{tr}, \phi) \in \mathsf{trace}(K_A)$, there exists $(\mathsf{tr}', \phi') \in \mathsf{trace}(K_B)$ such that $\mathsf{tr} = \mathsf{tr}'$ and $\phi \sim \phi'$. Two configurations K_A and K_B are* trace equivalent*, denoted by $K_A \approx K_B$, if $K_A \sqsubseteq K_B$ and $K_B \sqsubseteq K_A$.*

Note that only observable actions are taken into account in the definition of equivalence between two traces. A protocol (P, ϕ_0) is *trace included* in another protocol (Q, ψ_0), denoted by $(P, \phi_0) \sqsubseteq_t (Q, \psi_0)$, if the initial configuration of (P, ϕ_0) is included in the initial configuration of (Q, ψ_0). Two protocols (P, ϕ_0) and (Q, ψ_0) are trace equivalent, denoted by $(P, \phi_0) \approx_t (Q, \psi_0)$, if their initial configurations are trace equivalent.

Example 4. Consider the protocol $(P_A, \emptyset)$ of Example 2, and a variant $(P_B, \emptyset)$ where every occurrence of id_A is replaced by another public constant, id_B, and k_{AS} by k_{BS}. This new protocol serves the same purpose as the previous one, only executed by an agent with distinct identity. Ideally, for the sake of privacy, one could hope than using our passport does not disclose the

identity of its bearer. Formally, one would expect that $(P_A, \emptyset) \approx_t (P_B, \emptyset)$. Unfortunately, let us show that $(P_A, \emptyset) \not\approx_t (P_B, \emptyset)$. Let tr be the execution of $(P_A, \emptyset)$ described in Example 3. We need to show there exists no execution tr' of $(P_B, \emptyset)$ which shares the same visible labels and such that the final frames of tr and tr' are statically equivalent.

Because of the existence of only one branch using each channel, there is only one execution tr' matching the labels on the transitions of tr:

$$\mathsf{tr}' = K_0 \xrightarrow{\tau} \xrightarrow{\tau} \xrightarrow{\tau} \xrightarrow{\mathsf{in}(c_A^A, \mathsf{start})} \xrightarrow{\mathsf{out}(c_A^A, x_1)} \xrightarrow{\tau} \xrightarrow{\mathsf{in}(c_A^S, x_1)} \xrightarrow{\mathsf{out}(c_A^S, x_2)} \xrightarrow{\tau} \xrightarrow{\mathsf{in}(c_S, x_1)} \xrightarrow{\mathsf{out}(c_S, x_3)} (\emptyset; \mathcal{P}_B; \lambda)$$

where:

$$\begin{cases} K_0 = (\emptyset; \{P_B\}; \emptyset), \text{ the initial configuration of } (P_B, \emptyset) \\ \lambda = \{x_1 \triangleright \mathsf{senc}(id_B, k_{BS}), x_2 \triangleright h(id_B), x_3 \triangleright \mathsf{senc}(id_B, k_{\mathsf{auth}})\} \end{cases}$$

But, considering the recipes $M = h(id_A)$ and $N = x_2$, it is immediate that $\sigma \not\sim_{\mathsf{E}} \lambda$. Hence $(P_A, \emptyset) \not\approx_t (P_B, \emptyset)$.

Generally, the problem of deciding whether two protocols are trace-equivalent is harder than deciding the reachability of a term.

Theorem 1. *If the trace equivalence of two protocols is decidable, then the reachability problem is decidable.*

Proof. Suppose the trace equivalence of two protocols is decidable and let u be a term whose reachability in a protocol (P, ϕ_0) has to be decided. Let then $P_1 = P|\mathsf{in}(c, u).\mathsf{out}(c, \mathsf{tag}_1)$ and $P_2 = P|\mathsf{in}(c, u).\mathsf{out}(c, \mathsf{tag}_2)$ two processes; where c is a channel not used in P and tag_1, tag_2 are two public constants. Then u is reachable in (P, ϕ_0) if, and only if, $(P_1, \phi_0) \not\approx_t (P_2, \phi_0)$. Hence if trace equivalence is decidable, so is reachability. □

4 Decidability of trace equivalence

We now intend to define two classes of protocols for which we present results regarding the decidability of the trace equivalence of two protocols. As noted earlier, if the trace equivalence is quite a recent concept introduced with privacy-type properties, the reachability problem has been thoroughly studied. In particular, reachability has been shown to be decidable for a fragment $\mathcal{C}$ of first-order logic with clauses containing at most one variable [13], plus the needed clauses to express the attacker abilities. As Theorem 1 makes the trace equivalence problem harder than reachability, it is natural to interest ourselves in restrictions of the said fragment $\mathcal{C}$.

4.1 Classes of protocols

We propose first a class C_1 of protocols somewhat included in the fragment $\mathcal{C}$. Similarly, processes in C_1 have at most one variable and thus able only to copy one part of inputted message blindly.

For the sake of concision, $\epsilon!$ denotes either the presence or absence of a bang in front of a process.

Definition 4. *C_1 is the class of protocols (P, σ_0)*

$$P = \bigmid_{i=1}^{n} \epsilon!\mathsf{in}(c_i, u_i).\mathsf{out}(c_i, u_i')$$

such that: $\forall i \in \{1, \ldots, n\}$, $u_i, u_i' \in \mathcal{T}'(\Sigma, \{x\})$ where $\mathcal{T}'(\Sigma, \{x\})$ is the set of terms such that the right argument of senc *or* sdec *is always a constant of Σ,* i.e. *encryption with atomic and non-variable keys.*

Upon translation into Horn clauses [8], protocols of C_1 would fall into $\mathcal{C}$ if it were for the absence of bangs in front of some parallel branches, making the translation only approximate.

We also propose another, more constrained subclass of protocols, C_2, adding restrictions on the use of keys, in particular on the possibility for a protocol to disclose them and on the use of channels. Those restrictions will enable us to state the results of Section 4.3.

Definition 5. *C_2 is the class of protocols (P, σ_0) of C_1 such that:*

- *any constant used as a key in P must either be in σ_0 or not be reachable.*
- $\forall i, j \in \{1, \ldots, n\}, c_i = c_j \Rightarrow i = j$

In particular note that two distinct parallel branches in a process in C_2 cannot share the same channel.

4.2 Undecidability of C_1

Theorem 2 (undecidability of trace equivalence in C_1). *Trace equivalence in C_1 is undecidable.*

To prove the undecidability of C_1 with regards to trace equivalence, we show it is possible to encode the Post Correspondence Problem into an equivalence 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, they appear to be equivalent. On the other hand, if a solution to the Post Correspondence

Problem does exit, it will lead the second process to react in a distinct way, breaking the trace equivalence property.

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

$$\forall i \in \{1,\dots,n\},\ \begin{cases} u_i^* = i\cdot\alpha_1\cdot\ldots\cdot i\cdot\alpha_{|u_i|} \text{ where } u_i = \alpha_1\cdot\ldots\cdot\alpha_{|u_i|} \\ V_i = \{j_1\cdot\alpha_1\cdot\ldots\cdot j_{|v_i|}\cdot\alpha_{|v_i|} | v_i = \alpha_1\cdot\ldots\cdot\alpha_{|v_i|} \\ \qquad\qquad \wedge 1\leq k\leq |v_i| \Rightarrow 1\leq j_k\leq n\} \\ W_i = (A\cup\{1,\dots,n\})^{2|v_i|}\setminus V_i \\ W_i' = \bigcup_{k=1}^{2|v_i|-1}(A\cup\{1,\dots,n\})^k \end{cases}$$

Consider the signature with symmetric encryption enriched with constants for the letters in A and values in $\{1,\dots,n\}$. and two additional constants ϵ (for the empty word) and $\mathsf{success}$.

$$\begin{aligned} &\Sigma_{\mathsf{pub}} = \{\epsilon/0, \mathsf{success}/0, \mathsf{senc}/2, \mathsf{sdec}/2\}\cup\{a/0 | a\in A\}\cup\{1/0,\dots,n/0\} \\ &\Sigma_{\mathsf{prv}} = \{k_i, k_i'\}_{i\in\{0,\dots,3\}}\cup\{k, k_P, k_Q\} \\ &\Sigma = \Sigma_{\mathsf{pub}}\uplus\Sigma_{\mathsf{prv}} \end{aligned}$$

Concatenation on the right for word will be represented by encryption.

Let $P_U(k)$ and $P_V(k)$ be the following parametrised processes with only one variable x.

$$\begin{aligned} P_U(k) := \ & !\mathsf{in}(c_i, \mathsf{senc}(\mathsf{senc}(x\cdot u_i^*, k_0), k)).\mathsf{out}(c_i, \mathsf{senc}(\mathsf{senc}(x, k_1), k)) && (2)\\ | \ & !\mathsf{in}(c_i, \mathsf{senc}(\mathsf{senc}(x\cdot u_i^*, k_1), k)).\mathsf{out}(c_i, \mathsf{senc}(\mathsf{senc}(x, k_1), k)) && (3)\\ | \ & !\mathsf{in}(c', \mathsf{senc}(\mathsf{senc}(\epsilon, k_1), k)).\mathsf{out}(c', \mathsf{senc}(\mathsf{senc}(\epsilon, k_2), k)) && (4) \end{aligned}$$

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

$$\begin{aligned} P_V(k) := \ & !\mathsf{in}(c_i, \mathsf{senc}(\mathsf{senc}(x\cdot v, k_0'), k)).\mathsf{out}(c_i, \mathsf{senc}(\mathsf{senc}(x, k_1'), k)) && (2')\\ | \ & !\mathsf{in}(c_i, \mathsf{senc}(\mathsf{senc}(x\cdot v, k_1'), k)).\mathsf{out}(c_i, \mathsf{senc}(\mathsf{senc}(x, k_1'), k)) && (3')\\ | \ & !\mathsf{in}(c_i, \mathsf{senc}(\mathsf{senc}(x\cdot w, k_0'), k)).\mathsf{out}(c_i, \mathsf{senc}(\mathsf{senc}(\mathsf{senc}(x\cdot w, k), k_3'), k)) && (2'a)\\ | \ & !\mathsf{in}(c_i, \mathsf{senc}(\mathsf{senc}(x\cdot w, k_1'), k)).\mathsf{out}(c_i, \mathsf{senc}(\mathsf{senc}(\mathsf{senc}(x\cdot w, k), k_3'), k)) && (3'a)\\ | \ & !\mathsf{in}(c_i, \mathsf{senc}(\mathsf{senc}(w', k_0'), k)).\mathsf{out}(c_i, \mathsf{senc}(\mathsf{senc}(\mathsf{senc}(w', k), k_3'), k)) && (2'b)\\ | \ & !\mathsf{in}(c_i, \mathsf{senc}(\mathsf{senc}(w', k_1'), k)).\mathsf{out}(c_i, \mathsf{senc}(\mathsf{senc}(\mathsf{senc}(w', k), k_3'), k)) && (3'b)\\ | \ & !\mathsf{in}(c_i, \mathsf{senc}(\mathsf{senc}(x, k_3'), k)).\mathsf{out}(c_i, \mathsf{senc}(\mathsf{senc}(\mathsf{senc}(x, k), k_3'), k)) && (5')\\ | \ & !\mathsf{in}(c', \mathsf{senc}(\mathsf{senc}(\epsilon, k_1'), k)).\mathsf{out}(c', \mathsf{success}) && (4')\\ | \ & !\mathsf{in}(c', \mathsf{senc}(\mathsf{senc}(x\cdot\alpha, k_1'), k)).\mathsf{out}(c', \mathsf{senc}(\mathsf{senc}(x\cdot\alpha, k_2'), k)) && (4'a)\\ | \ & !\mathsf{in}(c', \mathsf{senc}(\mathsf{senc}(x, k_3'), k)).\mathsf{out}(c', \mathsf{senc}(\mathsf{senc}(x\rangle, k_2'), k)) && (4'b) \end{aligned}$$

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

We can now define P and Q to be:

$$\begin{aligned} P &:= \mathsf{in}(c_0, x).\mathsf{out}(c_0, \mathsf{senc}(\mathsf{senc}(x, k_0), k_P))|P_U(k_P) \\ Q &:= \mathsf{in}(c_0, x).\mathsf{out}(c_0, \mathsf{senc}(\mathsf{senc}(x, k'_0), k_Q))|P_V(k_Q) \end{aligned}$$

$(P, \emptyset)$ and $(Q, \emptyset)$ are indeed two protocols of C_1.

Example 5. Consider the following instance of the Post Correspondence Problem and let U and V be the following sequences of tiles built over the alphabet $A = \{a, b\}$:

$$\begin{aligned} U &= (b, aab, aab) \\ V &= (ba, ab, aaa) \end{aligned}$$

This instance has a solution, namely the sequence of indices $(1, 2)$ corresponding to the word $u = baab$. In our formalism, it leads to the following sets of terms, representing words over $(A \cdot \{1,2,3\})^*$:

$$\begin{array}{lll} u_1^* = 1b & u_2^* = 2a2a2b & u_3^* = 3a3a3b \\ V_1 = \{i_1 b i_2 a | i_1, i_2 \in \{1,2,3\}\} & & V_2 = \{i_3 a i_4 b | i_3, i_4 \in \{1,2,3\}\} \\ V_3 = \{i_5 a i_6 a i_7 a | i_5, i_6, i_7 \in \{1,2,3\}\} & & W_1 = (A \cup \{1,2,3\})^2 \setminus V_1 \\ W_2 = (A \cup \{1,2,3\})^2 \setminus V_2 & & W_3 = (A \cup \{1,2,3\})^3 \setminus V_3 \\ W'_1 = W'_2 = \bigcup_{k=1}^{3} (A \cup \{1,2,3\})^k & & W'_3 = \bigcup_{k=1}^{5} (A \cup \{1,2,3\})^k \end{array}$$

where, for instance, $1b$ is a shorthand for $\mathsf{senc}(\mathsf{senc}(\epsilon, 1), b)$, and $(A \cup \{1,2,3\})^k$ here also represents the set of terms associated to the words of $(A \cup \{1,2,3\})^k$.

Thus we can define the term u^* to be $u^* = u_1^* u_2^* = 1b2a2a2b$ and illustrate the traces of P and Q associated to it.

$$\begin{aligned} \mathsf{tr} = (P, \emptyset) &\xRightarrow{\mathsf{in}(c,u^*)} \xRightarrow{\mathsf{out}(c,x_1)} \xRightarrow{\mathsf{in}(c_2,x_1)} \xRightarrow{\mathsf{out}(c_2,x_2)} \\ &\xRightarrow{\mathsf{in}(c_1,x_2)} \xRightarrow{\mathsf{out}(c_1,x_3)} \xRightarrow{\mathsf{in}(c',x_3)} \xRightarrow{\mathsf{out}(c',x_4)} (P', \phi_P) \\ \mathsf{tr}' = (Q, \emptyset) &\xRightarrow{\mathsf{in}(c,u^*)} \xRightarrow{\mathsf{out}(c,x_1)} \xRightarrow{\mathsf{in}(c_2,x_1)} \xRightarrow{\mathsf{out}(c_2,x_2)} \\ &\xRightarrow{\mathsf{in}(c_1,x_2)} \xRightarrow{\mathsf{out}(c_1,x_3)} \xRightarrow{\mathsf{in}(c',x_3)} \xRightarrow{\mathsf{out}(c',x_4)} (Q', \phi_Q) \end{aligned}$$

where

$$\begin{aligned} \phi_P = \{ & x_1 \triangleright \mathsf{senc}(\mathsf{senc}(u^*, k_0), k_P), x_2 \triangleright \mathsf{senc}(\mathsf{senc}(1b, k_1), k_P), \\ & x_3 \triangleright \mathsf{senc}(\mathsf{senc}(\epsilon, k_1), k_P), x_4 \triangleright \mathsf{senc}(\mathsf{senc}(\epsilon, k_2), k_P)\} \end{aligned}$$

and

$$\begin{aligned} \phi_Q = \{ & x_1 \triangleright \mathsf{senc}(\mathsf{senc}(u^*, k'_0), k_Q), x_2 \triangleright \mathsf{senc}(\mathsf{senc}(1b2a, k'_1), k_Q), \\ & x_3 \triangleright \mathsf{senc}(\mathsf{senc}(\epsilon, k'_1), k_Q), x_4 \triangleright \mathsf{success}\} \end{aligned}$$

Intuitively tr unstacks first the tile u_2^*, then u_1^*. Then the attacker, knowing $\mathsf{senc}(\mathsf{senc}(\epsilon, k_1), k_P)$ can finally trigger a communication on channel c'. Note that, as u^* embeds the sequence of indices in itself, the attacker cannot start by unstacking u_3^*, even though $u_2 = u_3 = aab$: this is made to prevent her from using an input in two different ways and causing unfortunate equalities to hold in the frame.

As equivalent traces have to match in terms of channels and input recipes and the pattern matching in Q is exclusive, tr' is at this stage the only possible candidate to be equivalent to tr: tr' unstacks first the tile $2a2b$ belonging to V_2, and then the tile $1b2a$ belonging to V_1. The communication on c' leads to the display of the tag $\mathsf{success}$, thus preventing tr' from being equivalent to tr.

Similarly inputs corresponding to words which are not solutions of this Post Correspondence Problem will lead to statically equivalent frames thanks to the encoding with the secret key k_Q ($\mathsf{success}$ never appears), and ill-formed inputs will cause P to be stuck while Q will still be able to follow.

Proposition 1 (undecidability of trace inclusion). $(P, \emptyset) \sqsubseteq_t (Q, \emptyset)$ *if, and only if, PCP has no solution.*

To obtain a result on the trace equivalence of C_1, let us enrich our signature with two new private constants k_a and k_b. Let σ_0 be the empty frame; P' and Q' be:

$$\begin{aligned} P' := \;& \mathsf{in}(c_0, x).\mathsf{out}(c_0, \mathsf{senc}(\mathsf{senc}(x, k_0), k_P)) \\ & | \; \mathsf{in}(c'_0, x).\mathsf{out}(c'_0, \mathsf{senc}(k_a, k)) \\ & | \; \mathsf{in}(c'_0, x).\mathsf{out}(c'_0, \mathsf{senc}(k_b, k)) \\ & | \; \mathsf{in}(c''_0, \mathsf{senc}(x, k)).\mathsf{out}(c''_0, x) \\ & | \; P_U^a(k_P) \\ & | \; P_V^b(k'_P) \end{aligned}$$

and

$$\begin{aligned} Q' := \;& \mathsf{in}(c_0, x).\mathsf{out}(c_0, \mathsf{senc}(\mathsf{senc}(x, k'_0), k_Q)) \\ & | \; \mathsf{in}(c'_0, x).\mathsf{out}(c'_0, \mathsf{senc}(k_a, k)) \\ & | \; \mathsf{in}(c'_0, x).\mathsf{out}(c'_0, \mathsf{senc}(k_b, k)) \\ & | \; \mathsf{in}(c''_0, \mathsf{senc}(x, k)).\mathsf{out}(c''_0, x) \\ & | \; P_V^a(k_Q) \\ & | \; P_V^b(k'_Q) \end{aligned}$$

where for any process R, R^a (resp. R^b) is obtained by replacing every occurrence of $\mathsf{in}(c, u)$ by $\mathsf{in}(c, \mathsf{senc}(u, k_a))$ (resp. by $\mathsf{in}(c, \mathsf{senc}(u, k_b))$) for any c and any term u. (P', σ_0) and (Q', σ_0) are both protocols of C_1.

Proposition 2 (encoding of PCP). $(P', \sigma_0) \approx_t (Q', \sigma_0)$ *if, and only if,* PCP *has no solution.*

Theorem 2 (undecidability of trace equivalence in C_1). *Trace equivalence in* C_1 *is undecidable.*

Proof. It is possible to encode any instance of the Post Correspondence Problem into the problem of deciding whether two protocols are trace equivalence according to Proposition 2. □

4.3 Decidability of C_2

Decidability of trace equivalent, in spite of the strong constraints on C_2 is not trivial. Let us consider the following examples of non-equivalent protocols.

Example 6. Let a be a public constant in our signature Σ, k a private key, $\sigma_0 = \{x_1 \triangleright \mathsf{senc}^{2n}(a, k)\}$ and $\tau_0 = \{x_1 \triangleright \mathsf{senc}^{2n+1}(a, k)\}$ be two initial frames where $n \in \mathbb{N}$ and $\mathsf{senc}^n(u, k)$ is the n-th iterate of $\mathsf{senc}(\_, k)$ to u; and P be the following process:

$$P = !\mathsf{in}(c, x).\mathsf{out}(c, \mathsf{senc}(\mathsf{senc}(x, k), k))$$

Then $(P, \sigma_0) \not\approx_t (P, \tau_0)$. Indeed, (P, σ_0) is able to produce the term $\mathsf{senc}^{2n}(a, k)$, already present in σ_0, in n steps, whereas, because of the parity, (P, τ_0) is unable to match this equality.

A similar phenomenon can also be observed without actually requiring non-empty initial frames.

Example 7. Let Σ be the same signature defined in Example 6, P and Q be the processes

$$\begin{aligned} P = \;& !\mathsf{in}(c_1, x).\mathsf{out}(c_1, \mathsf{senc}^n(x, k)) && (1) \\ & \mid !\mathsf{in}(c_2, x).\mathsf{out}(c_2, \mathsf{senc}^p(x, k)) && (2) \end{aligned}$$

and

$$\begin{aligned} Q = \;& !\mathsf{in}(c_1, x).\mathsf{out}(c_1, \mathsf{senc}^n(x, k)) && (1') \\ & \mid !\mathsf{in}(c_2, x).\mathsf{out}(c_2, \mathsf{senc}^q(x, k)) && (2') \end{aligned}$$

where $n, p, q \in \mathbb{N}$. Then $(P, \emptyset) \not\approx_t (Q, \emptyset)$ as soon as $p \neq q$. Once again, $(P, \emptyset)$, by playing the branch (1) p times and (2) n times will cause an equality to hold, whereas, if $q \neq p$, playing $(2')$ n times will not result in an equality. And symmetrically by switching the roles of p and q.

These two example illustrate the need for a sufficiently refined method to deal with trace equivalence in C_2 and motivate the following results.

Theorem 3 (decidability of trace equivalence in C_2). *Trace equivalence in C_2 is decidable.*

We introduce a new technical class of protocols, C_2', whose signature is unary. The Lemma 1 enables us to focus rather on the equivalence problem of C_2'. To prove the decidability of the equivalence of two protocols P and Q in C_2', we go through a number of intermediate steps. We first reduce the problem of trace equivalence to the equivalence of two particular languages on words. This first reduction is achieved by Proposition 3. The main point here is to let words represent recipes leading to equalities within a frame. Hence an accepting word symbolises a particular equality when a non-accepting represents either an ill-formed recipe or an inequality.

Then we prove the existence of a generalised pushdown automaton recognising the said language in Proposition 4. The condition on distinct channels in the definition of C_2' ensures the determinism of the automaton which enables us to use a result from Sénizergues [21] to conclude about the decidability of the equivalence of two pushdown automata, and hence of two protocols in C_2'.

As we do not deal with nonces, we will use substitutions like frames as they do not contain any restricted names. For the sake of clarity, the initial frames σ_0 and τ_0 have domains $\{y_{1\dagger}, \ldots, y_{N\dagger}\}$ to avoid useless arithmetic on indices. Outputs still use variables x_i.

For this purpose, we introduce a new class S' of signatures Σ' of the form

$$\begin{array}{l}
\Sigma' = \Sigma'_{\mathsf{pub}} \uplus \Sigma'_{\mathsf{prv}} \\
\Sigma'_{\mathsf{pub}} = \{e_k/1, d_k/1\}_{k \in K_{\mathsf{pub}}} \cup K_{\mathsf{pub}} \cup H'_{\mathsf{pub}} \cup A'_{\mathsf{pub}} \\
\Sigma'^{+}_{\mathsf{pub}} = \{e_k/1\}_{k \in K_{\mathsf{pub}}} \qquad \Sigma'^{-}_{\mathsf{pub}} = \{d_k/1\}_{k \in K_{\mathsf{pub}}} \\
\Sigma'_{\mathsf{prv}} = \{e_k/1, d_k/1\}_{k \in K_{\mathsf{prv}}} \cup K_{\mathsf{prv}} \cup H'_{\mathsf{prv}} \cup A'_{\mathsf{prv}}
\end{array}$$

with the equational theory E':

$$E' = \{d_k(e_k(x)) = x | k \in K_{\mathsf{pub}} \cup K_{\mathsf{prv}}\}$$

where H_{pub} and H_{prv} (resp. A_{pub} and A_{prv}) are set of public and private unary function symbols (resp. public and private constants); K_{pub} and K_{prv} are sets of particular public and private constants used as keys and e_k (resp. d_k) represents symmetric encryption with constant k (resp. decryption with k). Finally, $\Sigma'^{+}_{\mathsf{pub}}$ is the set of public constructors, while $\Sigma'^{-}_{\mathsf{pub}}$ gathers the public destructors of Σ'.

Definition 6. *C_2' is the class of protocols (P, σ_0) on a signature $\Sigma' \in S'$ along with its equational theory,*

$$P = \bigmid_{i=1}^{n} \epsilon!\mathsf{in}(c_i, u_i).\mathsf{out}(c_i, u_i')$$

such that:

- $\forall i \in \{1, \ldots, n\}$, $u_i, u_i' \in \mathcal{T}(\Sigma', \{x\})$
- $\forall i, i' \in \{1, \ldots, n\}$, $i \neq i' \Rightarrow c_i \neq c_i'$

Lemma 1 (reduction from C_2 to C_2'). *Let (P, σ_0) and (Q, τ_0) be two protocols of C_2 on a signature $\Sigma \in S$. There exist a signature $\Sigma' \in S'$, two protocols (P', σ_0') and (Q', τ_0') of C_2' such that $(P, \sigma_0) \approx_t (Q, \tau_0)$ if, and only if, $(P', \sigma_0') \approx_t (Q', \tau_0')$.*

Note that the signature Σ' depends on the protocols P and Q: in particular it has to include as constants the keys used in these protocols.

Definition 7. *Let $P = \bigmid_{i=1}^{n} !\mathsf{in}(c_i, u_i).\mathsf{out}(c_i, u_i')$ be a process in C_2' with initial frame σ_0 and w a word in $\{1^\dagger, \ldots, N^\dagger\}.(\{1, \ldots, n\} \cup \Sigma'_{\mathsf{pub}})^*$. We define term_w^P by induction:*

- $\mathsf{term}_i^P = y_i\sigma_0$ *if* $i \in \{1^\dagger, \ldots, N^\dagger\}$
- $\mathsf{term}_{w \cdot i}^P = u_i'\theta$ *if* $i \in \{1, \ldots, n\}$ *and* $\theta = mgu(\mathsf{term}_w^P, u_j) \neq \bot$
- $\mathsf{term}_{w \cdot i}^P = (i.\mathsf{term}_w^P)\downarrow$ *if* $i \in \Sigma'_{\mathsf{pub}}$
- term_w^P *is otherwise undefined.*

The term term_w^P represents the interpretation of a word w, informally a linear recipe in our calculus, in the protocol (P, σ_0). Especially, in Proposition 3 we are interested in showing that an equality between two recipes in two executions of two protocols is actually equivalent to the equality of the "interpretations" of two words, getting us closer to pushdown automata.

Proposition 3 (language equivalence). *Let (P, σ_0) and (Q, τ_0) be two protocols of C_2, $\mathcal{L}_P$ and $\mathcal{L}_Q$ be the languages*

$$\mathcal{L}_P = \{i_1 \dots i_l \# j_m \dots j_1 | \mathsf{term}^P_{i_1 \dots i_l} = \mathsf{term}^P_{j_1 \dots j_m}\}$$

and

$$\mathcal{L}_Q = \{i_1 \dots i_l \# j_m \dots j_1 | \mathsf{term}^Q_{i_1 \dots i_l} = \mathsf{term}^Q_{j_1 \dots j_m}\}$$

in

$$\{1^\dagger, \dots, N^\dagger\}.(\{1, \dots, n\} \cup \Sigma'_{\mathsf{pub}})^*.\#.(\{1, \dots, n\} \cup \Sigma'_{\mathsf{pub}})^*.\{1^\dagger, \dots, N^\dagger\}$$

Then $(P, \sigma_0) \approx_t (Q, \tau_0)$ if, and only if, $\mathcal{L}_P = \mathcal{L}_Q$.

Definition 8 (GDPA). *A Generalised Deterministic Pushdown Automaton (GDPA) $\mathcal{A}$ can be defined as a 7-uple:*

$$\mathcal{A} = (Q, \Pi, \Gamma, q_0, \omega, A, \delta)$$

where

$$\begin{cases} Q \text{ is set of states} \\ \Pi \text{ is a finite set of input symbols} \\ \Gamma \text{ is a finite set of stack symbols} \\ q_0 \in Q \text{ is the initial state} \\ \omega \in \Gamma \text{ is the starting stack symbol} \\ A \subseteq Q \text{ is the set of accepting states} \\ \delta : (Q \times (\Pi \cup \{\epsilon\}) \times \Gamma^*) \to \mathcal{P}(Q \times \Gamma^*) \text{ is a transition function} \end{cases}$$

which moreover satisfies the following conditions:

- *For any $q \in Q$, $a \in \Pi \cup \{\epsilon\}$, $x \in \Gamma^*$, the set $\bigcup_{y \preceq x} \delta(q, a, y)$, where $\preceq$ is the prefix relation, has at most one element.*
- *For any $q \in Q$, $x \in \Gamma^*$, if $\delta(q, \epsilon, x) \neq \emptyset$, then $\delta(q, a, x) = \emptyset$ for every $a \in \Pi$.*

GDPA differ from traditional deterministic pushdown automata (DPA) as they can unstack several symbols at a time. A GPDA can easily be converted into a PDA by adding new states and ϵ-transitions.

Proposition 4 (existence of a GDPA). *There exists a GDPA $\mathcal{A}_P$ recognising $\mathcal{L}_P$.*

Proposition 5 (decidability of trace equivalence in C'_2). *Trace equivalence in C'_2 is decidable.*

Proof. Let P and Q be two protocols of C_2'. By Proposition 3, the trace-equivalence of P and Q is equivalent to the equality of the languages $\mathcal{L}_P$ and $\mathcal{L}_Q$. Thanks to Proposition 4, the equality of these two languages is itself reducible to the equivalence of two GDPA $\mathcal{A}_P$ and $\mathcal{A}_Q$. As GDPA can easily be transformed into deterministic pushdown automata, their equivalence is also decidable according to the result of Sénizergues in [21].

□

A direct application from Proposition 5 and Lemma 1 leads to the desired result.

Theorem 3 (decidability of trace equivalence in C_2). *Trace equivalence in C_2 is decidable.*

5 Conclusion

We have defined in this report a class of protocols with one variable only and without nonces for which the question of trace equivalence is shown to be undecidable through an encoding of the Post Correspondence Problem. Subsequently, we refined our class to achieve a decidability result relying on a reduction to the equivalence of two deterministic pushdown automata. As future work, it would be interesting to enrich our decidable fragment with other primitives, pairing in particular. Another possible trail lies on the weakening of the condition according to which parallel branches need to have distinct channels. It seems reasonable to aim for a looser condition involving exclusive pattern matching on the inputted and outputted terms. A longer term perspective would be to focus on a variation of our decidability result which would allow a effective procedure to be implemented, or to apply our approach to tagged protocols, such as described in Blanchet and Podelski [9] to obtain a new decidable class of protocols with regards to trace equivalence.

References

1. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proc. 28th Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115. ACM Press, 2001.
2. M. Abadi and A. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proc. 4th Conference on Computer and Communications Security (CCS'97)*, pages 36–47. ACM Press, 1997.
3. R. M. Amadio and W. Charatonik. On name generation and set-based analysis in the dolev-yao model. In *CONCUR*, pages 499–514, 2002.

4. M. Arapinis, T. Chothia, E. Ritter, and M. Ryan. Analysing unlinkability and anonymity using the applied pi calculus. In *Proc. 23rd Computer Security Foundations Symposium (CSF'10)*, pages 107–121. IEEE Computer Society Press, 2010.
5. A. Armando, R. Carbone, L. Compagna, J. Cuéllar, and M. L. Tobarra. Formal analysis of SAML 2.0 web browser single sign-on: breaking the SAML-based single sign-on for google apps. In *Proc. of the 6th ACM Workshop on Formal Methods in Security Engineering (FMSE 2008)*, pages 1–10. ACM, 2008.
6. A. Armando et al. The AVISPA Tool for the automated validation of internet security protocols and applications. In *Proc. 17th International Conference on Computer Aided Verification, CAV'2005*, volume 3576 of *LNCS*. Springer, 2005.
7. M. Baudet. Deciding security of protocols against off-line guessing attacks. In *ACM Conference on Computer and Communications Security*, pages 16–25, 2005.
8. B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proc. 14th Computer Security Foundations Workshop (CSFW'01)*. IEEE Comp. Soc. Press, 2001.
9. B. Blanchet and A. Podelski. Verification of cryptographic protocols: tagging enforces termination. *Theor. Comput. Sci.*, 333(1-2):67–90, 2005.
10. M. Bruso, K. Chatzikokolakis, and J. den Hartog. Formal verification of privacy for RFID systems. In *Proc. 23rd Computer Security Foundations Symposium (CSF'10)*. IEEE Computer Society Press, 2010.
11. V. Cheval, H. Comon-Lundh, and S. Delaune. Trace equivalence decision: negative tests and non-determinism. In *ACM Conference on Computer and Communications Security*, pages 321–330, 2011.
12. H. Comon, V. Cortier, and J. Mitchell. Tree automata with one memory, set constraints, and ping-pong protocols. In *ICALP*, pages 682–693, 2001.
13. H. Comon-Lundh and V. Cortier. New decidability results for fragments of first-order logic and application to cryptographic protocols. In *Proc. 14th Int. Conf. on Rewriting Techniques and Applications (RTA'2003)*, volume 2706 of *LNCS*, pages 148–164. Springer-Verlag, 2003.
14. C. Cremers. The Scyther Tool: Verification, falsification, and analysis of security protocols. In *Proc. 20th International Conference on Computer Aided Verification (CAV 2008)*, volume 5123/2008 of *LNCS*, pages 414–418. Springer, 2008.
15. S. Delaune, S. Kremer, and M. D. Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, (4):435–487, July 2008.
16. S. Delaune, S. Kremer, and M. D. Ryan. Symbolic bisimulation for the applied pi calculus. *Journal of Computer Security*, 18(2):317–377, 2010.
17. D. Dolev and A. C. Yao. On the security of public key protocols. In *Proc. 22nd Symposium on Foundations of Computer Science (FCS'81)*, pages 350–357. IEEE Computer Society Press, 1981.
18. N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Proc. Workshop on Formal Methods and Security Protocols*, 1999.
19. S. Even and O. Goldreich. On the security of multi-party ping-pong protocols. In *FOCS*, pages 34–39, 1983.
20. R. Ramanujam and S. P. Suresh. Tagging makes secrecy decidable with unbounded nonces as well. In *FSTTCS*, pages 363–374, 2003.
21. G. Sénizergues. L(a)=l(b)? decidability results from complete formal systems. *Theor. Comput. Sci.*, 251(1-2):1–166, 2001.
22. C. Stirling. Decidability of dpda equivalence. *Theor. Comput. Sci.*, 255(1-2):1–31, 2001.

A Appendix

Proposition 1 (undecidability of trace inclusion). *$(P, \emptyset) \sqsubseteq_t (Q, \emptyset)$ if, and only if, PCP has no solution.*

Proof (sketch). We want to prove successively the two implications :

1. *If PCP has a solution then $P \not\sqsubseteq_t Q$:*
   If PCP has a solution, there exists a word $u = \alpha_1 \dots \alpha_m \in A^+$, $p \in \mathbb{N}$ and $(i_k)_{0 \leq k \leq p} \in \mathbb{N}^p$ such that $u = u_{i_1} \cdot \dots \cdot u_{i_p} = v_{i_1} \cdot \dots \cdot v_{i_p}$. From this word and sequence, the attacker can build the term u^* representing the word $i_1 \cdot \alpha_1 \cdot \dots \cdot i_p \cdot \alpha_m$ from the tiles u_i^*. Let tr be the trace of P following the sequence (i_k):

   $$\mathsf{tr} = (P, \emptyset) \xrightarrow{\mathsf{in}(c,u^*)} \xrightarrow{\mathsf{out}(c,x_1)} \xrightarrow{\mathsf{in}(c_{i_p},x_1)} \xrightarrow{\mathsf{out}(c_{i_p},x_2)}$$
   $$\dots \xrightarrow{\mathsf{out}(c_{i_1},x_{p+1})} \xrightarrow{\mathsf{in}(c',x_{p+1})} \xrightarrow{\mathsf{out}(c',x_{p+2})} (P', \phi_P)$$

   tr model the fact that, given u^*, P can remove one by one the tiles $u_{i_p}^*$ to $u_{i_1}^*$ to reach the empty word and hence output the message $\mathsf{senc}(\mathsf{senc}(\epsilon, k_2), k_P)$. In this execution, the only input recipes are variables of the frame x_k and no equality holds in ϕ', as the attacker ignores the secret key k and all outputted message are different; thus all messages look like fresh values to her.
   We claim that there exists no equivalent trace in Q. Indeed, as the pattern matching operated by process parts $(2')$, $(3')$, $(2'a)$, $(3'a)$ and $(2'b)$, $(3'b)$ is exclusive, given u^* as an input (which is necessary as it is sent in clear), Q has no choice but to remove tiles $v_{i_p} \in V_{i_p}$ to $v_{i_1} \in V_{i_1}$ and output $\mathsf{success}$ on channel c' as u is a Post word. Any other trace would either lead to a mismatch on the channels or an improper filtering in Q. Then the recipe $x_{p+2} = \mathsf{success}$ is evaluated to true in ϕ_Q (the frame resulting from this execution of Q) but false in ϕ_P. So tr has no equivalent trace in Q: $P \not\sqsubseteq_t Q$.
2. *If PCP has no solution then $P \sqsubseteq_t Q$:*
   If PCP has no solution, then for every word $u^* \in (A \cup \{1, \dots, n\})^+$ either u^* cannot decomposed as a sequence of tiles u_i^* or u^* cannot decomposed as a sequence of tiles in V_i. Given a trace tr of P and the final frame ϕ_P, we claim that there exists a trace equivalent trace tr' of Q with final frame ϕ_Q.
   - **If u^* is not a word of $(A \cdot \{1, \dots, n\})^+$:** then P will not be able to progress but by unstacking the potential last tiles of u^*: as Q is always able to follow tr either with the tiles $v \in V_i$ or with the

branches $(2'a)$, $(3'a)$ and $(5')$ and because no equality other than those due to the replays of old messages holds in ϕ_P and ϕ_Q, tr is trace-equivalent to tr'.

- **If u^* has no decomposition in tiles u_i^*:** then P never outputs on channel c' and ϕ_P contains nothing but trivial equalities *i.e.* equalities between variables, consequences of replays operated by the attacker. tr' can be built by following the sequence of channels used in tr and choosing the adequate filtering ($(2')$, $(3')$, $(2'a)$, $(3'a)$, $(2'b)$ or $(3'c)$). As for ϕ_P, ϕ_Q will not contain any non-trivial equality as messages outputted are similar to random values to the attacker.
- **If u^* has a decomposition in tiles u_i^*,** and incidentally outputs on channel c': u^* cannot be decomposed in tiles of V_i with the same sequence of indices: because the filtering in Q is also exhaustive, messages outputted by Q (up to replays) from a certain point will be tagged by the constant 3 (*i.e.* of the form $\mathsf{senc}(\mathsf{senc}(\_, k_3'), k_Q)$), which will enable Q, through the part $(5')$, to match inputs and outputs on any channel c_i. Finally, parts $(4'a)$ and $(4'b)$ allow Q to match the outputs of P on c'. For the same reason as before, ϕ_P and ϕ_Q are statically equivalent as the messages appears to be random to the attacker and thus contain no non-trivial equalities.

Hence, for all traces of P there exists an equivalent trace of Q: $P \sqsubseteq_t Q$.

□

Proposition 2 (encoding of PCP). $(P', \sigma_0) \approx_t (Q', \sigma_0)$ *if, and only if,* PCP *has no solution.*

Proof (sketch).

- **If PCP has a solution, then $(P', \sigma_0) \not\approx_t (Q', \sigma_0)$:** Indeed any trace of (P', σ_0) will either result in an execution of $P_U^a(k_P)$ or $P_V^b(k_P')$; the result in the first case is a consequence of Theorem 1, thus witnessing the non-equivalence of the two protocols.
- **If PCP has no solution, then $(P', \sigma_0) \approx_t (Q', \sigma_0)$:** first, note that $(Q', \sigma_0) \sqsubseteq_t (P', \sigma_0)$ holds as the non-determinism in P' enables us to always choose an execution following P_V^b, and the reverse inclusion is a consequence also of Theorem 1 (if the execution uses P_U^a) or obvious (if it uses P_V^b).

□

Lemma 1 (reduction from C_2 to C_2'). *Let (P, σ_0) and (Q, τ_0) be two protocols of C_2 on a signature $\Sigma \in S$. There exist a signature $\Sigma' \in S'$, two protocols (P', σ_0') and (Q', τ_0') of C_2' such that $(P, \sigma_0) \approx_t (Q, \tau_0)$ if, and only if, $(P', \sigma_0') \approx_t (Q', \tau_0')$.*

Proof. Let (P, σ_0) and (Q, τ_0) be two protocols of C_2. We first explain how to build the protocol $\Sigma' \in S'$, (P', σ_0') in C_2', and symmetrically for (Q', τ_0). We will then prove that if $(P', \sigma_0') \not\approx_t (Q', \tau_0')$ then $(P, \sigma_0) \not\approx_t (Q, \tau_0)$; and conclude by proving the opposite implication.

Defining Σ': By Definition 5, the keys used in P and Q must either be non reachable or present in their respective initial frames: consequently, there exist two sets of indices I and J of respectively σ_0 and τ_0 such that for all $i \in I$ and $j \in J$, $x_i\sigma_0$ and $x_j\tau_0$ are the keys known to the attacker. Three cases occur:

- if for all $i, i' \in I \cup J$, $x_i = x_{i'}$ holds in σ_0 if, and only if it holds in τ_0 and moreover if all terms $x_i\sigma_0$ and $x_i\tau_0$ are constants of Σ: in that case, we can rename every $x_i\tau_0$ occuring in (Q, τ_0) by $x_i\sigma_0$. This renaming still preserves the relation between (P, σ_0) and (Q, τ_0).
- if for all $i, i' \in I \cup J$, $x_i = x_{i'}$ holds in σ_0 if, and only if it holds in τ_0 but there is a term $x_i\sigma_0$ or $x_i\tau_0$ which is not a constant of Σ: then the two processes are not equivalent. Indeed, Definition 5 forbids non-atomic encryption and thus one of the two terms $\mathsf{senc}(x_i, x_i)\sigma_0$ or $\mathsf{senc}(x_i, x_i)\tau_0$ is an element in $\mathcal{T}'(\Sigma, \{x\})$, but not the other. In that case, rename the non-atomic term with a new public constant: the renamed protocol and the other are still non-equivalent.
- else there exists $i, i' \in I \cup J$ such that, *e.g.* $x_i = x_{i'}$ holds in σ_0 but not in τ_0: then, (P, σ_0) and (Q, τ_0) are *a fortiori* not trace equivalent. In that case, no renaming is necessary.

Now, the protocol (P, σ_0) and the (possibly renamed) protocol (Q, τ_0) share the same set of disclosed keys $K_{\mathsf{pub}} = \{x_i\sigma_0 | i \in I\} \cup \{x_j\tau_0 | j \in J\}$ (the union need not be disjoint if there were a renaming). We can also define K_{prv} to be the set of constants in (P, σ_0) and (Q, τ_0) used as keys but not reachable. This preliminary work is indeed necessary to ensure that we can define a common signature Σ' for both protocols:

$$\begin{aligned}
\Sigma' &= \Sigma'_{\mathsf{pub}} \uplus \Sigma'_{\mathsf{prv}} \\
\Sigma'_{\mathsf{pub}} &= \{e_k/1, d_k/1\}_{k \in K_{\mathsf{pub}}} \cup K_{\mathsf{pub}} \cup H_{\mathsf{pub}} \cup (A_{\mathsf{pub}} \setminus K_{\mathsf{pub}}) \\
\Sigma'_{\mathsf{prv}} &= \{e_k/1, d_k/1\}_{k \in K_{\mathsf{prv}}} \cup K_{\mathsf{prv}} \cup H_{\mathsf{prv}} \cup (A_{\mathsf{prv}} \setminus K_{\mathsf{prv}})
\end{aligned}$$

with the equational theory E':

$$E' = \{d_k(e_k(x)) = x | k \in K_{\mathsf{pub}} \cup K_{\mathsf{prv}}\}$$

Defining (P', σ_0')*:* We can now define σ_0' to be the substitution built from σ_0 and P' to be the process built from P where every occurrence of $\mathsf{senc}(\_, k)$ has been replaced by e_k, and every occurrence of $\mathsf{sdec}(\_, k)$ has been replaced by d_k. By definition of Σ' and C_2 then: $(P', \sigma_0') \in C_2'$. We similarly define (Q', τ_0').

In this proof and the following ones, we only interest ourselves in *respectful* executions, *i.e.* traces where every input is immediately followed by its corresponding output. It is obvious that there is a witness for non-equivalence if, and only if, there exists a witness which is respectful: missing outputs need just to be inserted at the appropriate positions, and the frame to be re-indexed. We prove successively the two implications.

Suppose first that $(P', \sigma_0') \not\approx_t (Q', \tau_0')$*:* let tr' be an execution of (P', σ_0') such that there exists no equivalent execution of (Q', τ_0'). Build tr the execution of (P, σ_0) from tr', where every occurrence of $e_k(\_)$ in the labels, for any $k \in K$, is replaced by $\mathsf{senc}(\_, x_i)$, where i is the index of σ_0' such that $x_i\sigma_0' = k$. Because keys occurring in the recipes (and thus the labels) have to be reachable, by Definition 5, they necessarily appear in σ_0' and σ_0 for a certain (identical) index. Similarly every occurrence of $e_k(\_)$ in the frames of tr' is replaced by $\mathsf{senc}(\_, k)$. And symmetrically for $\mathsf{sdec}(\_, k)$ and $d_k(\_)$. tr is then an execution of (P, σ_0) (note that indices in the frame are identical in σ_0 and σ_0'). We claim that there exists no equivalent execution of (Q, τ_0). Indeed, if there were one, one could invert the previous process to retrieve an execution of (Q', τ_0') and contradicts our hypothesis. Hence $(P, \sigma_0) \not\approx_t (Q, \tau_0)$.

Suppose now that $(P, \sigma_0) \not\approx_t (Q, \tau_0)$*:* let tr be an execution of (P, σ_0) such that there exists no equivalent execution of (Q, τ_0). One can revert the construction used for the previous implication: from tr, let us build an execution tr' of (P', σ_0') as follows. In the labels, every occurrence of $k \in K$ in a recipe can be replaced by a variable x_i for some index i by Definition 5. One can then replace every occurrence of $\mathsf{senc}(\_, x_i)$ in the labels by $e_k(\_)$. In the frames, every occurrence of $\mathsf{senc}(\_, k)$ can be replaced by $e_k(\_)$. We proceed symmetrically for $\mathsf{sdec}(\_, k)$ and $d_k(\_)$. Similarly, tr' is then a valid execution of (P, σ_0). Moreover, we claim that there exists no equivalent execution of (Q', τ_0'): if there were one, one could invert the previous process to get back an execution of (Q, τ_0) and contradicts our hypothesis. Hence $(P', \sigma_0) \not\approx_t (Q', \tau_0)$.

And thus, deciding trace equivalence in C_2 comes down to deciding trace equivalence in C_2'. □

Proposition 3 (language equivalence). *Let (P, σ_0) and (Q, τ_0) be two protocols of C_2, $\mathcal{L}_P$ and $\mathcal{L}_Q$ be the languages*

$$\mathcal{L}_P = \{i_1 \dots i_l \# j_m \dots j_1 | \mathsf{term}^P_{i_1 \dots i_l} = \mathsf{term}^P_{j_1 \dots j_m}\}$$

and

$$\mathcal{L}_Q = \{i_1 \dots i_l \# j_m \dots j_1 | \mathsf{term}^Q_{i_1 \dots i_l} = \mathsf{term}^Q_{j_1 \dots j_m}\}$$

in

$$\{1^\dagger, \dots, N^\dagger\}.(\{1, \dots, n\} \cup \Sigma'_{\mathsf{pub}})^*.\#.(\{1, \dots, n\} \cup \Sigma'_{\mathsf{pub}})^*.\{1^\dagger, \dots, N^\dagger\}$$

Then $(P, \sigma_0) \approx_t (Q, \tau_0)$ if, and only if, $\mathcal{L}_P = \mathcal{L}_Q$.

Proof. – **Suppose $\mathcal{L}_P \neq \mathcal{L}_Q$:** for instance, let $w = i_1 \dots i_l \# j_m \dots j_1 \in \mathcal{L}_P \setminus \mathcal{L}_Q$. Let $i_{k_1}, \dots, i_{k_I}, j_{k'_1}, \dots, j_{k'_J}$ ($k_1 < \dots < k_I$ and $k'_1 < \dots < k'_J$) the letters of w in $\{1, \dots, n\}$. For every $a \in \{1, \dots, I\}$, we define R_a to be the recipe:

- $R_1 = i_{k_1 - 1} \dots i_2.y_{i_1}$ (if $a = 1$)
- $R_a = i_{k_a - 1} \dots i_{k_{a-1}+1}.x_{k_{a-1}}$

and similarly, for every $b \in \{1, \dots, J\}$, we define R'_b to be the recipe:

- $R'_1 = j_{k'_1 - 1} \dots j_2.y_{j_1}$ (if $b = 1$)
- $R'_b = i_{k'_b - 1} \dots j_{k'_{b-1}+1}.x_{k_{b-1}}$

and

$$\begin{aligned}
\mathsf{tr} = (P, \sigma_0) &\xrightarrow{\mathsf{in}(c_{i_{k_1}}, R_1)} (P_1, \sigma_0) \xrightarrow{\mathsf{out}(c_{i_{k_1}}, x_1)} (P'_1, \sigma_1) \xrightarrow{\mathsf{in}(c_{i_{k_2}}, R_2)} \dots \\
&\dots \xrightarrow{\mathsf{in}(c_{i_{k_I}}, R_I)} (P_I, \sigma_{I-1}) \xrightarrow{\mathsf{out}(c_{i_{k_I}}, x_I)} (P'_I, \sigma_I) \\
&\xrightarrow{\mathsf{in}(c_{j_{k'_1}}, R'_1)} (P_{I+1}, \sigma_I) \xrightarrow{\mathsf{out}(c_{j_{k'_1}}, x_{I+1})} \dots \\
&\dots \xrightarrow{\mathsf{in}(c_{j_{k'_J}}, R'_J)} (P_{I+J}, \sigma_{I+J-1}) \xrightarrow{\mathsf{out}(c_{j_{k'_J}}, x_{I+J})} (P'_{I+J}, \sigma_{I+J})
\end{aligned}$$

In particular, tr is a valid trace of P such that the equality

$$i_l(\dots(i_{k_I+1}(x_I))) \downarrow = (j_m \dots (j_{k_J+1}(x_{I+J})) \downarrow$$

holds in σ_{I+J} as $w \in \mathcal{L}_P$.
From our semantics and the definition of term^P we get that $i_l(\dots(i_{k_I+1}(x_I)))\sigma_{I+J} \downarrow = \mathsf{term}^P_{i_1 \dots i_l}$ and $\mathsf{term}^P_{j_1 \dots j_m} = (j_m \dots (j_{k_J+1}(x_{I+J}))\sigma_{I+J} \downarrow$. Moreover by definition of $\mathcal{L}_P$, $\mathsf{term}^P_{i_1 \dots i_l} = \mathsf{term}^P_{j_1 \dots j_m}$. On the other hand, because of the exclusive input pattern matching and the absence of nonces, given tr, either there exits no trace tr' of Q matching the labels of tr: in that case

the equivalence fails immediately; or there exists only one trace tr' of Q matching the labels of tr: the restriction on respectful executions forbid the process to accumulate inputs before outputting and thus creating non-determinism. Given the symmetric notation, we hence have that the same equality does not hold in σ'_{I+J} as $w \notin \mathcal{L}_Q$ and by definition of $\mathcal{L}_Q$, $\mathsf{term}^Q_{i_1 \dots i_l} \neq \mathsf{term}^Q_{j_1 \dots j_m}$. Consequently, $i_l(\dots(i_{k_I+1}(x_I)))\sigma'_{I+J}\downarrow \neq (j_m \dots (j_{k_J+1}(x_{I+J}))\sigma'_{I+J}\downarrow$.

– **Suppose** $P \not\approx_t Q$**:** for instance, suppose $P \not\sqsubseteq_t Q$ *i.e.* there exists tr a trace of P such that for every trace tr' of Q, tr is not equivalent to tr'. Because we can focus on respectful executions of the processes, given tr, there exists at most one trace tr' of Q with adequate labels (respectful executions forces the process to output as soon as he receives an input, and thus behaving in a deterministic way). We make the following case analysis:

1. *There exists exactly one trace* tr' *of* Q *matching the labels of* tr. We can then rephrase the hypothesis $P \not\sqsubseteq_t Q$ in an equivalent way: there exists a respectful trace tr of P with final frame σ, two recipes R and R' such that $R\sigma\downarrow = R'\sigma\downarrow$ but $R\sigma'\downarrow \neq R'\sigma'\downarrow$; or $R\sigma\downarrow \neq R'\sigma\downarrow$ but $R\sigma'\downarrow = R'\sigma'\downarrow$, where σ' is the final frame of the unique trace tr' of Q which matches the labels of tr.
   As R (resp. R') is a recipe built on a unary public signature, it is of the form $s^1_1 \dots s^1_{k_1}.x_{l_1}$ or $s^1_1 \dots s^1_{k_1}.y_{l_1}$ (if the term is built from the initial frame σ_0) where for all $i \in \{1,\dots,k_1\}$, $s^j_i \in \Sigma'_{\mathsf{pub}}$ (resp. $s'^1_1 \dots s'^1_{k'_1}.x_{l'_1}$ or $s'^1_1 \dots s'^1_{k'_1}.y_{l'_1}$). Each variable appearing in R (resp. R') was either in the initial frame (and then is a y_i variable) or appears first in an input and output triggered by the attacker on a channel c_{n_1} through a recipe R_1 (resp. on a channel $c_{n'_1}$ through a recipe R'_1).
   One can then build a sequence $l_1.s_{k_1} \dots s_1$ (if it was a y_i variable) or $n_1.s^1_{k_1} \dots s^1_1$ (if it were the result of outputting on channel c_{n_1}). In the latter case, the input was itself the result of applying a linear recipe to a variable x_{l_2} or y_{l_2}. Because the execution and the recipes are finite, we can inductively build a sequence by adding as a prefix the reversed recipe which triggered the input and the index of the channel. This sequence w would be of the form

$$w = l_1.s^p_{k_p} \dots s^p_1.n_p \dots n_1 s^1_{k^1_1} \dots s^1_1$$

   where $l_1 \in \{1^\dagger,\dots,N^\dagger\}$, $p \in \mathbb{N}$, for all $j \in \{1,\dots,p\}$ and $i \in \{1,\dots,k_p\}$, $s^j_i \in \Sigma'_{\mathsf{pub}}$; and $n_i \in \{1,\dots,n\}$. And symmetrically we would get a sequence w' from R'. One possess then w (resp. w') of

symbols of $\{1, \ldots, n\}$ (the channels) and Σ'_{pub} (the recipes used by the attacker) and starting by a element of $\{1^\dagger, \ldots, N^\dagger\}$ (as terms in σ have to be closed, the attacker must start with an element of σ_0). Let then W be the word $W = w\#w'^R$ where $.^R$ denotes the reverse (mirror) operation. In particular, because of the adequacy between our definition of term and the semantics, we get that $R\sigma\downarrow = \mathsf{term}^P_w$, $R'\sigma\downarrow = \mathsf{term}^P_{w'}$ and symmetrically $R\sigma'\downarrow = \mathsf{term}^Q_w$, $R'\sigma'\downarrow = \mathsf{term}^Q_{w'}$ for tr' is a trace of Q with the same labels as tr so term^Q_w and term^Q_w are indeed defined. Hence, as the equality must hold in σ or σ' but not both, either $W \in \mathcal{L}_P \setminus \mathcal{L}_Q$ (the equality holds in P) or $W \in \mathcal{L}_Q \setminus \mathcal{L}_P$ (the equality holds in Q). So $\mathcal{L}_P \neq \mathcal{L}_Q$ as expected.

2. *There exists no trace* tr' *of* Q *matching the labels of* tr.
Following the same process as in the previous case, we build a word $W = w\#w^R \in \mathcal{L}_P$ witnessing the equality $x_{|\sigma|}\sigma = x_{|\sigma|}\sigma$. On the other hand, as term^Q_w and $\mathsf{term}^Q_{w'}$ are undefined (consequence of the absence of matching trace), we get that $W \notin \mathcal{L}_Q$, proving that $\mathcal{L}_P \neq \mathcal{L}_Q$.

□

Proposition 4 (existence of a GDPA). *There exists a GDPA $\mathcal{A}_P$ recognising $\mathcal{L}_P$.*

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

We extend our working definition of GDPA with an additional transition labelled flush, whose effect is to erase the entire content of the stack (except the starting stack symbol). Such a transition can easily be emulated with an additional state and ϵ-transitions. We denote by $q \xrightarrow{a;u/\mathsf{flush}/v} q'$ a transition form q to q' reading a, popping u from the stack, flushing it and finally pushing v.

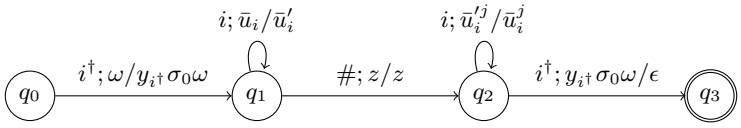

Fig. 2. Automaton $\mathcal{A}_P$

Proof. We introduce here some notations: given a term u in $\mathcal{T}(\Sigma', \{x\})$ we define $\bar{u}$ to be the context such that $u = \bar{u}[x]$. As Σ' contains only unary symbols and constants, $\bar{u}$ can be interpreted as a word in Σ'^*. Similarly, given a ground term u, we consider it to both an element of $\mathcal{T}(\Sigma')$ and Σ'^*.

We will also refer to the current stack of a run in the automaton by stack (or stack_i during a execution). Once again, from our stack alphabet, it is possible to interpret a stack as a term in $\mathcal{T}(\Sigma', \{x\})$: letters from Σ' are viewed as constant or unary function symbols, the letter x as the variable x and the starting stack symbol is ignored. In particular substitutions can be applied: $\mathsf{stack}\sigma$ denotes the result of applying σ to the interpretation of stack as a term.

The relation $\prec$ represents the strict prefix relation for words in Γ^*. Note that the bottom of the stack, where the starting stack symbol lies most of the time, is on the right hand of the stack (seen as a word) and hence a prefix can be seen as its head.

We treat first the case where every parallel branch is preceded by a bang: let $\mathcal{A}_P$ be the (extended) GDPA schematically defined in Figure 2:

$$\mathcal{A}_P = (\{q_0, q_1, q_2, q_3\}, \Pi, \Gamma, q_0, \omega, \{q_3\}, \delta)$$

where
$\begin{cases} \Pi = \{1^\dagger, \dots, N^\dagger\} \cup \{1, \dots, n\} \cup \Sigma'_{\mathsf{pub}} \cup \{\#\} \\ \Gamma = \Sigma' \cup \{\omega, x\} \end{cases}$
and δ is defined as follows:

- **From q_0 to q_1:** for every $i^\dagger \in \{1^\dagger, \dots, N^\dagger\}$ there exists a transition $(i^\dagger; \omega / y_{i^\dagger}\sigma_0\omega)$.
- **From q_1 to q_1:**
  - for every $e \in \Sigma'^+_{\mathsf{pub}}$ and $z \in \Gamma \setminus \{\omega\}$, there exists a transition $(e; z/ez)$.
  - for every $d \in \Sigma'^-_{\mathsf{pub}}$ and $z \in \Gamma \setminus \{\omega, e\}$ such that $\{d(e(x)) = x\} \subseteq \mathsf{E}$, there exist a transition $(d; z/dz)$, and a transition $(d; e/\epsilon)$.
  - for every $i \in \{1, \dots, n\}$ such that u'_i *is not ground*, there exists a transition $(i; \bar{u}_i/\bar{u}'_i)$.
  - for every $i \in \{1, \dots, n\}$ such that u'_i *is ground*, there exists a transition $(i; \bar{u}_i/\mathsf{flush}/\bar{u}'_i)$.
- **From q_1 to q_2:** for every $z \in \Gamma \setminus \{\omega\}$ there exists a transition $(\#; z/z)$.
- **From q_2 to q_2:**
  - for every $e \in \Sigma'^+_{\mathsf{pub}}$ and $z \in \Gamma \setminus \{\omega\}$, there exists a transition $(e; e/\epsilon)$.
  - for every $d \in \Sigma'^-_{\mathsf{pub}}$ and $z \in \Gamma \setminus \{\omega, d\}$ such that $\{d(e(x)) = x\} \subseteq \mathsf{E}$, there exist a transition $(d; z/ez)$, and a transition $(d; d/\epsilon)$.
  - for every $i \in \{1, \dots, n\}$ such that u'_i *is not ground*, for every $\bar{u}' \prec \bar{u}'_i$, there exist a transition $(i; \bar{u}'.x.\omega/\bar{u}_i.x.\omega)$, and a transition $(i; \bar{u}'_i/\bar{u}_i)$.

- for every $i \in \{1,\dots,n\}$ such that u'_i *is ground*, for every $\bar{u}' \prec \bar{u}'_i$, there exist a transition $(i; \bar{u}'.x.\omega / \bar{u}_i.x.\omega)$ and a transition $(i; \bar{u}'_i.\omega / \bar{u}_i.x.\omega)$.

– **From** q_2 **to** q_3**:** for every $i^\dagger \in \{1,\dots,N^\dagger\}$ and every $\bar{u} \prec y_{i^\dagger}\sigma_0$, there exist a transition $(i^\dagger; \bar{u}.x.\omega/\epsilon)$ and a transition $(i^\dagger; y_{i^\dagger}\sigma_0.\omega/\epsilon)$

First, remark that, as protocols of C2 have the exclusive pattern matching property, the resulting automaton $\mathcal{A}_P$ is indeed deterministic. Then, it is easy to observe that necessarily $L(\mathcal{A}_P)$, the language recognized by $\mathcal{A}_P$, is contained in $\{1,\dots,N^\dagger\}.(\{1,\dots,n\} \cup \Sigma'_{\mathsf{pub}})^*.\#.(\{1,\dots,n\} \cup \Sigma'_{\mathsf{pub}})^*.\{1,\dots,N^\dagger\}$ as a transition from q_0 to q_1 or from q_2 to q_3 needs to read a letter in $\{1,\dots,N^\dagger\}$ and the transition from q_1 to q_2 is triggered by $\#$. So any word $w \in L(\mathcal{A}_P)$ can be written as $w = w_1 \# w_2^R$, where $w_1, w_2 \in \{1,\dots,N^\dagger\}.(\{1,\dots,n\} \cup \Sigma'_{\mathsf{pub}})^*$ and $w_1 = i1 \dots i_l, w_2 = j_1 \dots j_m$. We need to prove now that $w \in L(\mathcal{A}_P)$ if, and only if, $\mathsf{term}^P_{w_1} = \mathsf{term}^P_{w_2}$, according to the definition of $\mathcal{L}_P$.

1. **Suppose** $\mathsf{term}^P_{w_1} = \mathsf{term}^P_{w_2}$**:**
   (a) *Reading* w_1*:* the first transition from q_0 to q_1 is possible upon reading $i_1 \in \{1,\dots,N^\dagger\}$. After this, we get that $\mathsf{stack}_1 = y_{i_1}\sigma_0 = \mathsf{term}^P_{i_1}$ *(i)*. If for some $k \in \{1,\dots,l\}$, $\mathsf{stack}_k = term^P_{i1\dots i_k}$, then the transition $q_1 \xrightarrow{i_{k+1}} q_1$ (with appropriate popping/pushing words, if i_{k+1} is in $\{1,\dots,n\}$ or Σ'_{pub}) is the only possible one, and the definition of $\mathsf{term}^P_{i1\dots i_k}$ ensures to be able to pop the appropriate letters from Γ while the definition of $\mathsf{term}^P_{i1\dots i_{k+1}}$ ensures that $\mathsf{stack}_{k+1} = \mathsf{term}^P_{i1\dots i_{k+1}}$ *(ii)*. Finally, by induction from *(i)* and *(ii)* to l we get that w_1 can be read and after reading it, $\mathsf{stack}_l = \mathsf{term}^P_{w_1}$ *(iii)*.
   (b) *Reading* w_2^R*:* because $\mathsf{term}^P_{w_1} = \mathsf{term}^P_{w_2}$, after reading $\#$, $\mathsf{stack}'_m = \mathsf{term}^P_{w_2}$ as the stack is left unchanged by the transition from q_1 to q_2. Moreover we define $\theta_m = \emptyset$ (the empty substitution) *(i')*. Then if $\mathsf{stack}'_{k+1}\theta_{k+1} = term^P_{j1\dots j_{k+1}}$ then $q_2 \xrightarrow{j_k} q_2$ (or $q_2 \xrightarrow{j_1} q_3$ as $j_1 \in \{1,\dots,N^\dagger\}$) is the only possible transition. By defining θ_k accordingly, $\theta_k = mgu(u^q_{j_{k+1}}, \mathsf{term}^P_{j1\dots j_k})$ if the transition is $q_2 \xrightarrow{j_k; \bar{u}'.x.\omega/\bar{u}^q_{j_{k+1}}.x.\omega} q_2$ for instance, we ensure that $\mathsf{stack}'_k\theta_k = \mathsf{term}^P_{j1\dots j_k}$ *(ii')*. By induction from *(i')* and *(ii')* up to index i_2 we get that $\mathsf{stack}'_2\theta_2 = term^P_{j1.j2}$ in q_2: the transition from q_2 to q_3 is possible: $w \in L(\mathcal{A}_P)$.
2. **Suppose** $w \in L(\mathcal{A}_P)$**:**
   (a) *Reading* w_1*:* we follow the same induction as in point 1a. Transitions are known to be possible, as w is accepted, but we get to know that $\mathsf{stack}_l = \mathsf{term}^P_{w_1}$ after reading w_1.
   (b) *Reading* w_2^R*:* the process is similar to the point 1b but backwards. We define a sequence of substitution on w_2. First by $\phi_1 = mgu(\mathsf{stack}'_1, y_{j_1}\sigma_0)$,

where stack'_1 is the stack of $\mathcal{A}_P$ before reading j_1. This definition ensures that $\mathsf{stack}'_1\phi_1 = y_{j_1}\sigma_0 = \mathsf{term}^P_{i_1}$ *(i)*. For any transition $(q_2, \mathsf{stack}'_{k+1}, \phi_{k+1}) \xrightarrow{j_k} (q_2, \mathsf{stack}_k, \phi_k)$ along the path of w_2, we define ϕ_{k+1} accordingly. If $j_k \in \{1, \ldots, n\}$ for instance:

$$\phi_{k+1} = mgu(\mathsf{stack}'_{k+1}, u'^q_{j_k} mgu(\mathsf{stack}'_k\phi_k, u^q_{i_k}))$$

Because the transition is indeed possible, as w is accepted, the two mgu are defined and different from $\bot$. In particular, if $\mathsf{stack}'_k\phi_k = \mathsf{term}^P_{j_1 \ldots j_k}$ then $\mathsf{stack}'_{k+1}\phi_{k+1} = \mathsf{term}^P_{j_1 \ldots j_{k+1}}$ *(ii)*. By induction from *(i)* and *(ii)* until j_m, we get that $\mathsf{stack}'_m\phi_m = \mathsf{term}^P_{j1 \ldots j_m}$, and as stack'_m is ground, $\mathsf{stack}'_m = \mathsf{term}^P_{w_2}$. As the stacks are identical before and after reading #, we can conclude that $\mathsf{term}^P_{w_1} = \mathsf{term}^P_{w_2}$.

Hence we built an GDPA $\mathcal{A}_P$ whose language is exactly $\mathcal{L}_P$.

To extend this proof to the case where some parallel branches do not have a bang in front of them, we build a variation of the GDPA $\mathcal{A}_P$, $\mathcal{A}'_P$. Let $\mathcal{I}$ the subset of $\{1, \ldots, n\}$ gathering the branches of P without a bang. The states q_1 and q_2 in $\mathcal{A}_P$ are replaced by the states q^s_1 and q^s_2 for $s \in \mathcal{P}(\mathcal{I})$: transitions labelled with letters *not* in $\mathcal{I}$ are duplicated in every one of them whereas for every $i \in \mathcal{I}$ a transition $q_1 \xrightarrow{i} q_1$ (resp. $q_2 \xrightarrow{i} q$, where $q \in \{q_2, q_3\}$) is replaced by the transitions $q_1^{s \uplus \{i\}} \xrightarrow{i;z} q^s_1$ (resp. $q_2^{s \uplus \{i\}} \xrightarrow{i;z} q^s_2$; or $q_2^{s \uplus \{i\}} \xrightarrow{i;z} q_3$ if $q = q_3$) where z represents any popping/pushing combination. Then $\mathcal{A}'_P$ still recognizes exactly $\mathcal{L}_P$ and moreover the branches without bang are used at most once as expected from our semantics. □