From security protocols to pushdown automata ⋆

Rémy Chrétien[1,2], Véronique Cortier[1], and Stéphanie Delaune[2]

[1] LORIA, CNRS, France
[2] LSV, ENS Cachan & CNRS & INRIA Saclay Île-de-France

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

1 Introduction

Formal methods have been successfully applied for rigorously analyzing security protocols. In particular, many algorithms and tools (see [13, 4, 9, 2, 11] 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, an attacker should never learn a secret (secrecy property) or make 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 such. They are instead defined as indistinguishability properties in [1, 6]. 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.

⋆ Full version available at http://hal.inria.fr/hal-00817230. 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.

– Based on a procedure developed by M. Baudet [3], it has been shown that trace equivalence is decidable for deterministic processes with no else branches, and for a family of equational theories that captures most standard primitives [10]. A simplified proof of [3] has been proposed by Y. Chevalier and M. Rusinowitch [8].
– A. Tiu and J. Dawson [17] have designed and implemented a procedure for open bisimulation, a notion of equivalence stronger than the standard notion of trace equivalence. This procedure only works for a limited class of processes.
– V. Cheval *et al.* [7] have proposed and implemented a procedure for trace equivalence, and for a quite general class of processes. They consider non deterministic processes that use standard primitives, and that may involve else branches.

However, these decidability results analyse equivalence for a *bounded number of sessions* only, that is assuming that protocols are executed a limited number of times. This is of course a strong limitation. Even if no flaw is found when a protocol is executed n times, there is absolutely no guarantee that the protocol remains secure when it is executed $n+1$ times. And actually, the existing tools for a bounded number of sessions can only analyse protocols for a very limited number of sessions, typically 2 or 3. Another approach consists in implementing a procedure that is not guaranteed to terminate. This is in particular the case of ProVerif [4], a well-established tool for checking security of protocols. ProVerif is able to check equivalence although it does not always succeed [5]. Of course, Proverif does not correspond to any decidability result.

Our contribution. We study the decidability of equivalence of security protocols for an unbounded number of sessions. Even in the case of reachability properties such as secrecy, the problem is undecidable in general. We therefore focus on a class of protocols for which secrecy is decidable [9]. This class typically assumes that each protocol rule manipulates at most one variable. Surprisingly, even a fragment of this class (with only symmetric encryption) turns out to be undecidable for equivalence properties. We consequently further assume our protocols to be deterministic (that is, given an input, there is at most one possible output). We show that equivalence is decidable for an unbounded number of sessions and for protocols with standard primitives but pairs. 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 [14]. We actually characterize equivalence of protocols in terms of equivalence of deterministic generalized real-time pushdown automata, that is deterministic pushdown automata with no epsilon-transition but such that the automata may unstack several symbols at a time. More precisely, we show how to associate to a process P an automata $\mathcal{A}_P$ such that two processes are equivalent if, and only if, their corresponding automata yield the same language and, reciprocally, we show how to associate to an automata $\mathcal{A}$ a process $P_{\mathcal{A}}$ such that two automata yield the same language if, and only if, their corresponding processes are equivalent, that is:

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

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

2 Model for security protocols

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

2.1 Syntax

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

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

We further assume an infinite set Σ_0 of *constant symbols* of sort key or msg, an infinite set $\mathcal{C}h$ of constant symbols of sort channel, two infinite sets of *variables* $\mathcal{X}, \mathcal{W}$, and an infinite set $\mathcal{N} = \mathcal{N}_{\mathsf{pub}} \uplus \mathcal{N}_{\mathsf{prv}}$ of *names* of sort rand: $\mathcal{N}_{\mathsf{pub}}$ represents the random numbers drawn by the attacker while $\mathcal{N}_{\mathsf{prv}}$ represents the random numbers drawn by the protocol's participants. As usual, *terms* are defined as names, variables, and function symbols applied to other terms. We denote by $\mathcal{T}(\mathcal{F}, \mathcal{N}, \mathcal{X})$ the set of terms built on function symbols in $\mathcal{F}$, names in $\mathcal{N}$, and variables in $\mathcal{X}$. We simply write $\mathcal{T}(\mathcal{F}, \mathcal{N})$ when $\mathcal{X} = \emptyset$. We consider three particular signatures:

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

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

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

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

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

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

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

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

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

Sometimes, we will omit the null process. We write $fv(P)$ for the set of *free variables* that occur in P, *i.e.* the set of variables that are not in the scope of an input. A *protocol* is a ground process, *i.e.* a process P such that $fv(P) = \emptyset$.

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

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

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

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

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

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

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

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

2.2 Semantics

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

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

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

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

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

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

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

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

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

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

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

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

2.3 Trace equivalence

Intuitively, two processes are equivalent if they cannot be distinguished by any attacker. Trace equivalence can be used to formalise many interesting security properties, in particular privacy-type properties, such as those studied for instance in [1, 6]. We first introduce a notion of intruder's knowledge well-suited to cryptographic primitives for which the success of decrypting or checking a signature is visible.

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

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

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

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

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

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

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

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

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

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

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

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

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

3 Ping-pong protocols

We aim at providing a decidability result for the problem of trace equivalence between protocols in presence of replication. However, it is well-known that replication leads to undecidability even for the simple case of reachability properties. Thus, we consider a class of protocols, called $\mathcal{C}_{\mathsf{pp}}$, for which (in a slightly different setting), reachability has already been proved decidable [9].

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* u (resp. *output term* v) is a term defined by the grammars given below:

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

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

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

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

1. *for all $i \in \{1, \ldots, n\}$, and $j \in \{1, \ldots, p_i\}$, $k_j^i \in \mathbb{N}$, u_j^i is an input term, and v_j^i is an output term where names occurring in v_j^i are included in $\{r_1, \ldots, r_{k_j^i}\}$;*
2. *for all $i \in \{1, \ldots, n\}$, and $j_1, j_2 \in \{1, \ldots, p_i\}$, if $j_1 \neq j_2$ then for any renaming of variables, $u_{j_1}^i$ and $u_{j_2}^i$ are not unifiable*[3].

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

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

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

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

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

3.2 Undecidability results

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

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

This result is shown by encoding the Post Correspondence Problem (PCP). Alternatively, it results from the reduction result established in Section 5 and the undecidability result established in [12]. Undecidability of trace inclusion actually implies undecidability of trace equivalence as soon as processes are non deterministic. Indeed consider the choice operator $+$ whose (standard) semantics is given by the following rules:

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

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

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

4 From trace equivalence to language equivalence

This section is devoted to a sketch of proof of Theorem 1. Deciding trace equivalence is done in two main steps. First, we show how to reduce the trace equivalence problem between protocols in $\mathcal{C}_{\mathsf{pp}}$, to the problem of deciding trace equivalence (still between protocols in $\mathcal{C}_{\mathsf{pp}}$) when the attacker acts as a *forwarder*.

Then, we encode the problem of deciding trace equivalence for forwarding attackers into the problem of language equivalence for real-time generalized pushdown deterministic automata (GPDA).

4.1 Generalized pushdown automata

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

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

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

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

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

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

A real-time GPDA can easily be converted into a DPA by adding new states and ϵ-transitions. Thus, the problem of language equivalence for two real-time GPDA $\mathcal{A}_1$ and $\mathcal{A}_2$, i.e. deciding whether $\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2)$ is decidable [15].

4.2 Getting rid of the attacker

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

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

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

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

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

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

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

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

To maintain the equivalence, we do a similar transformation in both P and Q relying on the bijection α. We ensure that the set of deducible keys has been correctly guessed by adding of some extra processes. Then the main step of the proof consists in showing that the forwarder has now the same power as a full attacker, although he cannot reuse the same randomness in two distinct encryptions/signatures, as a real attacker could.

4.3 Encoding a protocol into a real-time GPDA

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

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

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

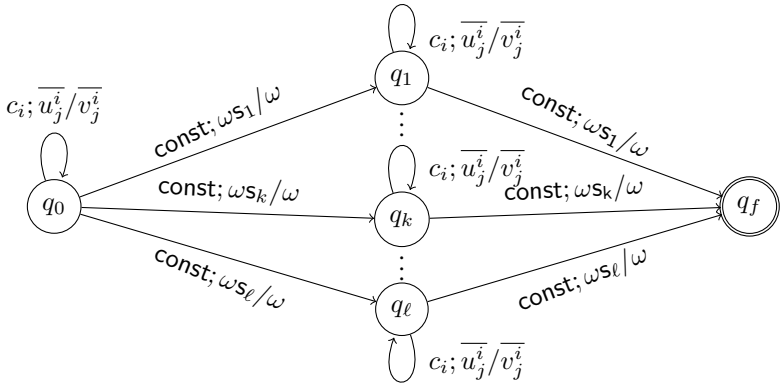
$$\bar{\mathsf{s}} = \mathsf{s} \text{ for any constant } \mathsf{s} \in \Sigma_0 \cup \{\mathsf{start}\}; \text{ and } \overline{\mathsf{f}(v, \mathsf{k}, r)} = \bar{v}.\mathsf{k} \text{ otherwise.}$$

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

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

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

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



5 From language equivalence to trace equivalence

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

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

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

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

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

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

Therefore, checking for equivalence of protocols is as difficult as checking equivalence of real-time generalized pushdown deterministic automata. It follows that the exact complexity of checking equivalence of protocols is unknown. The only upper bound is that equivalence is at most primitive recursive. This bound comes from the algorithm proposed by C. Stirling for equivalence of DPA [16] (Icalp 2002). Whether equivalence of DPA (or even real-time GPDA) is e.g. at least NP-hard is unknown.

6 Conclusion

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

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

to protocols with pair is not straightforward. A direction is to use pushdown automata for which stacks are terms.

G. Sénizergues is currently implementing his procedure for pushdown automata [14]. As soon as the tool will be available, we plan to implement our translation, yielding a tool for automatically checking equivalence of security protocols, for an unbounded number of sessions.

References

1. M. Arapinis, T. Chothia, E. Ritter, and M. Ryan. Analysing unlinkability and anonymity using the applied pi calculus. In *23rd Computer Security Foundations Symposium (CSF'10)*, pages 107–121. IEEE Computer Society Press, 2010.
2. D. Basin, S. Mödersheim, and L. Viganò. A symbolic model checker for security protocols. *Journal of Information Security*, 4(3):181–208, 2005.
3. M. Baudet. Deciding security of protocols against off-line guessing attacks. In *12th ACM Conference on Computer and Communications Security (CCS'05)*. ACM Press, 2005.
4. B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *14th Computer Security Foundations Workshop (CSFW'01)*. IEEE Computer Society Press, 2001.
5. B. Blanchet, M. Abadi, and C. Fournet. Automated Verification of Selected Equivalences for Security Protocols. In *20th Symposium on Logic in Computer Science*, 2005.
6. M. Bruso, K. Chatzikokolakis, and J. den Hartog. Formal verification of privacy for RFID systems. In *23rd Computer Security Foundations Symposium (CSF'10)*, 2010.
7. V. Cheval, H. Comon-Lundh, and S. Delaune. Trace equivalence decision: Negative tests and non-determinism. In *18th ACM Conference on Computer and Communications Security (CCS'11)*. ACM Press, 2011.
8. Y. Chevalier and M. Rusinowitch. Decidability of equivalence of symbolic derivations. *J. Autom. Reasoning*, 48(2):263–292, 2012.
9. H. Comon-Lundh and V. Cortier. 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)*, volume 2706 of *LNCS*. Springer, 2003.
10. V. Cortier and S. Delaune. A method for proving observational equivalence. In *22nd IEEE Computer Security Foundations Symposium (CSF'09)*. IEEE Computer Society Press, 2009.
11. C. Cremers. Unbounded verification, falsification, and characterization of security protocols by pattern refinement. In *15th ACM conference on Computer and communications security (CCS'08)*. ACM, 2008.
12. E. P. Friedman. The inclusion problem for simple languages. *Theor. Comput. Sci.*, 1(4):297–316, 1976.
13. M. Rusinowitch and M. Turuani. Protocol Insecurity with Finite Number of Sessions and Composed Keys is NP-complete. *Theoretical Computer Science*, 299:451–475, April 2003.
14. G. Sénizergues. The equivalence problem for deterministic pushdown automata is decidable. In *24th Int. Coll. on Automata, Languages and Programming (ICALP'97)*, LNCS, 1997.
15. G. Sénizergues. L(A)=L(B)? Decidability results from complete formal systems. *Theor. Comput. Sci.*, 251(1-2):1–166, 2001.
16. C. Stirling. Deciding DPDA equivalence is primitive recursive. In *29th International Colloquium on Automata, Languages and Programming ICALP'02*, LNCS. Springer, 2002.
17. A. Tiu and J. E. Dawson. Automating open bisimulation checking for the spi calculus. In *23rd IEEE Computer Security Foundations Symposium (CSF'10)*, pages 307–321, 2010.