POR for Security Protocol Equivalences*

Beyond Action-Determinism

David Baelde[1], Stéphanie Delaune[2], and Lucca Hirschi[3]

[1] LSV, ENS Paris-Saclay & CNRS, Inria Paris, Université Paris-Saclay, France
[2] Univ Rennes, CNRS, IRISA, France
[3] Department of Computer Science, ETH Zurich, Switzerland

Abstract. Formal methods have proved effective to automatically analyse protocols. Recently, much research has focused on verifying *trace equivalence* on protocols, which is notably used to model interesting *privacy* properties such as anonymity or unlinkability. Several tools for checking trace equivalence rely on a naive and expensive exploration of all interleavings of concurrent actions, which calls for partial-order reduction (POR) techniques. In this paper, we present the first POR technique for protocol equivalences that does not rely on an action-determinism assumption: we recast trace equivalence as a reachability problem, to which persistent and sleep set techniques can be applied, and we show how to effectively apply these results in the context of symbolic execution. We report on a prototype implementation, improving the tool DeepSec.

1 Introduction

Security protocols are notoriously difficult to design and their flaws can have a huge impact. Leaving aside implementation flaws and weaknesses of cryptographic primitives, there is already a long history of logical mistakes in the basic design of protocols, *e.g.*, [30,5,13,4]. At this level of detail, protocols can however be represented in the so-called symbolic model, which makes them amenable to automated formal verification. This approach has lead to mature tools and industrial successes, *e.g.*, [15,6,31].

Verification techniques have focused at first on reachability properties of protocols, used to model, *e.g.*, secrecy or authentication. More recently, equivalence properties have received a lot of attention, as they are often necessary to model privacy properties such as ballot secrecy in e-voting [25], anonymity or unlinkability [4,16]. Equivalence verification is complex, and each of the various state-of-the-art techniques has its own limitations. Tools for verifying scenarios with an unbounded number of sessions such as Proverif [15] or Tamarin [14] are usually efficient but only support a constrained form of equivalence, namely *diff-equivalence*, which is too limiting, *e.g.*, to model unlinkability [29]. Many tools

* This work has been partially supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No 714955-POPSTAR), as well as from the French National Research Agency (ANR) under the project TECAP.

for verifying bounded scenarios rely on symbolic execution [24]. For instance, Apte [18] and its successor DeepSec [21] implement an algorithm that explores all symbolic executions, maintaining pairs of sets of symbolic states and solving at each step complex equality, deducibility and indistinguishability constraints. Akiss [17] follows a different approach, enumerating all symbolic executions to check that none yields a non-equivalence witness. The strength of these tools is that they decide *trace equivalence*, which can adequately capture *e.g.*, unlinkability. However, their algorithms are very costly and, despite recent progress, it is still only possible to analyse small scenarios in a reasonable amount of time.

All the techniques mentioned above for deciding trace equivalence of security protocols rely on an enumeration of all symbolic executions including all interleavings of concurrent actions. This is obviously a cause of major inefficiency, which has lead to a quest for *partial-order reduction* (POR) techniques. These techniques, which have a long and successful history in traditional software verification [28,33,11], generally consist in leveraging action independencies to restrict the interleavings that a model-checking algorithm explores. In the context of verifying reachability properties for security protocols, some specific POR techniques have sometimes had to be devised [22,32], but there have also been successful uses of generic POR techniques such as *sleep sets* [23] (see [9] for a detailed discussion). In the context of verifying *trace equivalence* for security protocols, the only available POR techniques are, to the best of our knowledge, the ones we proposed in [8,9]. These *ad hoc* techniques have lead to significant performance gains in Apte and DeepSec [21,9]. However, they crucially rely on an *action-determinism* assumption (*i.e.*, once the observable trace is fixed, the system is deterministic) which is limiting in practice. For instance, there is no precise modeling of unlinkability involving action-deterministic systems.

In this paper, we present the first POR technique for checking trace equivalence on security protocols, without any action-determinism assumption. Our first step towards this goal is to recast the trace equivalence problem as a reachability problem in a carefully designed labeled transition system (LTS), to which we can then apply persistent and sleep set techniques. However, this result is not directly useful in practice, for several reasons. First, this LTS is infinitely branching, due to the arbitrary choices that the attacker can make when interacting with the protocol. This is the main issue addressed in protocol equivalence checkers, typically through symbolic execution. Second, determining when two actions are independent (the first ingredient of POR techniques) is far from obvious in our LTS. Independencies are often approximated through simple static checks in practical POR algorithms [28] but, as we shall see, it does not seem feasible in our setting without losing too many independencies. Instead, we determine independencies by exploring symbolic executions. We ignore constraint solving in that process, as it would be too expensive: this trade-off allows us to detect enough independencies at a reasonable cost. More generally, we show how to compute persistent sets in the same style, to eventually obtain a symbolic form of the sleep set technique. Third, the direct symbolic approach would still be overly expensive, due to another typical state explosion problem caused by con-

ditionals [12]. We circumvent it by showing that conditionals can be simplified, and often eliminated, in a way that does not affect persistent set computations. This approach yields a POR technique that is fast enough and allows to significantly reduce the number of symbolic traces to consider when checking trace equivalence. It is also independent of the specific verification algorithms that will be used to check equivalence along the reduced set of traces. We implemented the technique as a library to validate it experimentally.

Outline. We present a standard security protocol model in Section 2. After recalling persistent and sleep set techniques in Section 3, we design in Section 4 our concrete equivalence LTS to which they apply. Section 5 then defines a symbolic abstraction of this LTS, and shows how it can be used to obtain effective POR algorithms, notably through the collapse of conditionals. Finally, we present our implementations and experimental results in Section 6. Detailed proofs of all our results are available in [10].

2 Model for security protocols

We model security protocols in a variant of the applied pi-calculus [1]: processes exchange messages represented by terms quotiented by an equational theory.

2.1 Syntax

We assume a number of disjoint and infinite sets: a set $\mathcal{C}h$ of channels, denoted by c or d; a set $\mathcal{N}$ of *names*, denoted by n or k; a set $\mathcal{X}$ of *variables*, denoted by x or y; and a set $\mathcal{W}$ of *handles* of the form $\mathsf{w}_{c,i}$ with $c \in \mathcal{C}h$ and $i \in \mathbb{N}$, which will be used for referring to previously output terms. Next, we consider a *signature* Σ consisting of a set of function symbols together with their arity. Terms over a set of atomic data A, written $\mathcal{T}(A)$, are inductively generated from A and function symbols from Σ. When $A \subseteq \mathcal{N}$, elements of $\mathcal{T}(A)$ are called *messages* and written m. When $A \subseteq \mathcal{W}$, they are called *recipes* and written M, N. Intuitively, recipes express how a message has been derived by the environment (attacker) from the messages obtained so far. Finally, we consider an equational theory E over terms to assign a meaning to function symbols in Σ.

Protocols are then modelled through *processes* using the following grammar:

$$P, Q := 0 \mid \mathtt{in}(c, x).P \mid \mathtt{out}(c, u).P \mid \mathtt{if}\ u = v\ \mathtt{then}\ P\ \mathtt{else}\ Q \mid (P \mid Q) \mid P + Q$$

where $c \in \mathcal{C}h$, $u, v \in \mathcal{T}(\mathcal{N} \uplus \mathcal{X})$ and $x \in \mathcal{X}$. The process 0 does nothing. The process $\mathtt{in}(c, x).P$ expects a message m on the public channel c, and then behaves like $P\{x \mapsto m\}$, *i.e.*, P in which x has been replaced by m. The process $\mathtt{out}(c, u).P$ outputs u on the public channel c, and then behaves like P. We have constructions to perform tests (modulo E), parallel composition, and non-deterministic choice. We do not consider replication, and thus we do not need a specific "new" operation: we assume that names are implicitly freshly generated.

Example 1. We consider $\Sigma_{\mathsf{enc}} = \{\mathsf{enc}, \mathsf{dec}, \mathsf{mac}, \langle\ \rangle, \mathsf{proj}_1, \mathsf{proj}_2, \mathsf{nonce}_{\mathsf{err}}, \mathsf{mac}_{\mathsf{err}}\}$. The symbols enc, dec, and mac of arity 2 represent encryption, decryption and

message authentication code; concatenation of messages is modelled through the symbol $\langle\ \rangle$ of arity 2, with projection functions proj_1 and proj_2 of arity 1. The function symbols $\mathsf{nonce}_{\mathsf{err}}$ and $\mathsf{mac}_{\mathsf{err}}$ are constants (arity 0) that are used to model error messages. Then, we reflect the properties of the cryptographic primitives through the equational theory induced by the following equations:

$$\mathsf{dec}(\mathsf{enc}(x,y),y) = x, \quad \mathsf{proj}_1(\langle x,y\rangle) = x, \text{ and } \quad \mathsf{proj}_2(\langle x,y\rangle) = y.$$

We consider the BAC protocol used in e-passports which aims at establishing a fresh session key derived from k_P and k_R. Informally, we have:

1. $P \to R : n_P$
2. $R \to P : \mathsf{enc}(\langle n_R, \langle n_P, k_R\rangle\rangle, k_E),\ \mathsf{mac}(\mathsf{enc}(\langle n_R, \langle n_P, k_R\rangle\rangle, k_E), k_M)$
3. $P \to R : \mathsf{enc}(\langle n_P, \langle n_R, k_P\rangle\rangle, k_E),\ \mathsf{mac}(\mathsf{enc}(\langle n_P, \langle n_R, k_P\rangle\rangle, k_E), k_M)$

The keys k_E and k_M are long term keys shared between the passport P and the reader R. First, P sends a fresh random number n_P to the reader, and the reader answers to this challenge by generating its own nonce n_P, as well as k_R to contribute to a fresh session key. This encryption together with a mac is sent to the passport. The passport will then check the mac, decrypt the ciphertext and verify whether the nonce inside corresponds to the nonce n_P generated at the first step. In case decryption fails or the nonce inside the message is not the expected one, an error message will be sent. Otherwise, a message is sent to the reader. After checking that the message is the expected one, both entities are able to compute the fresh session key derived form k_R and k_P. In our syntax, we model the role of the passport as follows:

$$\begin{aligned} P(k_E,k_M) = {} & \mathtt{out}(c,n_P).\mathtt{in}(c,x).\\ & \mathtt{if}\ \mathsf{mac}(\mathsf{proj}_1(x),k_M) = \mathsf{proj}_2(x)\\ & \mathtt{then\ if}\ \mathsf{proj}_1(\mathsf{proj}_2(\mathsf{dec}(\mathsf{proj}_1(x),k_E))) = n_P\\ & \qquad \mathtt{then}\ \mathtt{out}(c,\langle m_P, \mathsf{mac}(m_P,k_M)\rangle).0\\ & \qquad \mathtt{else}\ \mathtt{out}(c,\mathsf{nonce}_{\mathsf{err}}).0\\ & \mathtt{else}\ \mathtt{out}(c,\mathsf{mac}_{\mathsf{err}}).0 \end{aligned}$$

where $m_P = \mathsf{enc}(\langle n_P, \langle \mathsf{proj}_1(\mathsf{dec}(\mathsf{proj}_1(x),k_E)), k_P\rangle\rangle, k_E)$.

2.2 Semantics

A *configuration* $\mathcal{K}$ is a pair $(\mathcal{P};\phi)$ where: $\mathcal{P}$ is either a multiset of processes with no free variable, or a special object $\bot_i$ with $i \in \mathbb{N}$; and $\phi = \{w_i \triangleright m_i\}_{1\le i\le n}$ is a *frame*, *i.e.*, a substitution of *domain* $\mathrm{dom}(\phi) = \{w_1,\ldots,w_n\} \subseteq \mathcal{W}$ such that the m_i are messages. Configurations $(\bot_i;\phi)$ are called *ghost configurations* dead at age i, and will only become useful in Section 4. Other configurations are said to be *alive*.

The operational semantics is given as an LTS on (alive) configurations, with the relation $\overset{\alpha}{\mapsto}$ defined in Figure 1. There, the index of the next output to be performed on channel c is defined as

$$\#_c(\mathrm{dom}(\phi)) = \max(\{0\} \cup \{j+1 \mid \mathsf{w}_{c,j} \in \mathrm{dom}(\phi)\}).$$

$$(\{\mathtt{in}(c,x).Q\} \uplus \mathcal{P};\phi) \xrightarrow{\mathtt{in}(c,M)} (\{Q\{x \mapsto M\phi\}\} \uplus \mathcal{P};\phi) \qquad \text{if } M \in \mathcal{T}(\mathrm{dom}(\phi))$$
$$(\{\mathtt{out}(c,u).Q\} \uplus \mathcal{P};\phi) \xrightarrow{\mathtt{out}(c,\mathsf{w}_{c,i})} (\{Q\} \uplus \mathcal{P};\phi \cup \{\mathsf{w}_{c,i} \triangleright u\}) \text{ with } i = \#_c(\mathrm{dom}(\phi))$$
$$(\{\mathtt{if}\ u = v\ \mathtt{then}\ Q_1\ \mathtt{else}\ Q_2\} \uplus \mathcal{P};\phi) \xmapsto{\tau} (\{Q_1\} \uplus \mathcal{P};\phi) \qquad \text{if } u =_{\mathsf{E}} v$$
$$(\{\mathtt{if}\ u = v\ \mathtt{then}\ Q_1\ \mathtt{else}\ Q_2\} \uplus \mathcal{P};\phi) \xmapsto{\tau} (\{Q_2\} \uplus \mathcal{P};\phi) \qquad \text{if } u \neq_{\mathsf{E}} v$$
$$(\{Q_1 + Q_2\} \uplus \mathcal{P};\phi) \xmapsto{\tau} (\{Q_1\} \uplus \mathcal{P};\phi) \qquad (\{Q_1 + Q_2\} \uplus \mathcal{P};\phi) \xmapsto{\tau} (\{Q_2\} \uplus \mathcal{P};\phi)$$
$$(\{Q_1 \mid Q_2\} \uplus \mathcal{P};\phi) \xmapsto{\tau} (\{Q_1, Q_2\} \uplus \mathcal{P};\phi) \qquad (\{0\} \uplus \mathcal{P};\phi) \xmapsto{\tau} (\mathcal{P};\phi)$$

Fig. 1. Operational semantics of processes

A process may input a term that an attacker built using the knowledge available to him through the frame, where messages output by the protocol are added. The output rule slightly differs from the standard one, which would use a fresh handle variable. Our use of fixed constants $\mathsf{w}_{c,i}$ makes it possible to view the transition system as a standard LTS, without any notion of freshness or α-renaming. Anticipating on the next sections where we build on top of this a different LTS encoding trace equivalence, we note that this design choice does not create spurious dependencies. We do not model internal communications, assuming instead that the attacker controls all communications (all channels are public). The last rules evaluate conditionals (modulo E), break parallel operators, remove null processes, and perform non-deterministic choices.

The relation $\mathcal{K} \xmapsto{\alpha_1 \dots \alpha_k} \mathcal{K}'$ between configurations, where $k \geq 0$ and each α_i is an observable or a τ action, is defined in the usual way. Given a sequence tr of actions, we denote $\mathsf{obs}(\mathsf{tr})$ the sequence of actions obtained by erasing τ actions.

Example 2. Let $\mathcal{K}_{\mathsf{same}} = (P(k_E, k_M); \phi_0)$ with $\phi_0 = \{\mathsf{w}_{c',0} \triangleright \langle m'_R, \mathsf{mac}(m'_R, k_M)\rangle\}$ and $m'_R = \mathsf{enc}(\langle n'_R, \langle n'_P, k'_R\rangle\rangle, k_E)$. Intuitively, the configuration $\mathcal{K}_{\mathsf{same}}$ represents a situation where the attacker initially knows part of a past transcript (*i.e.*, ϕ_0) of the passport under consideration (*i.e.*, $P(k_E, k_M)$). We have that

$$\mathcal{K}_{\mathsf{same}} \xmapsto{\mathtt{out}(c,\mathsf{w}_{c,0}).\mathtt{in}(c,\mathsf{w}_{c',0}).\tau.\tau.\mathtt{out}(c,\mathsf{w}_{c,1})} (0; \phi_0 \uplus \{\mathsf{w}_{c,0} \triangleright n_P; \mathsf{w}_{c,1} \triangleright \mathsf{nonce}_{\mathsf{err}}\}).$$

2.3 Equivalences

Many privacy-type properties (*e.g.*, ballot privacy in e-voting, unlinkability) are modelled relying on trace equivalence. In our setting, this behavioural equivalence relies on a notion of static equivalence that captures indistinguishable sequences of messages.

Definition 1. *Two frames ϕ and ψ are in* static equivalence*, $\phi \sim_s \psi$, when* $\mathrm{dom}(\phi) = \mathrm{dom}(\psi)$*, and $M\phi =_{\mathsf{E}} N\phi$ iff $M\psi =_{\mathsf{E}} N\psi$ for any $M, N \in \mathcal{T}(\mathrm{dom}(\phi))$.*

This equivalence is then lifted from sequences of messages to configuration.

Definition 2. *Let $\mathcal{K}_P = (\mathcal{P};\phi)$ and $\mathcal{K}_Q = (\mathcal{Q};\psi)$ be two configurations with $\mathrm{dom}(\phi) = \mathrm{dom}(\psi)$. We write $\mathcal{K}_P \sqsubseteq_t \mathcal{K}_Q$ if for every execution $\mathcal{K}_P \overset{\mathsf{tr}_1}{\longmapsto} (\mathcal{P}';\phi')$, there exists tr_2 and $(\mathcal{Q}';\psi')$ such that $\mathcal{K}_Q \overset{\mathsf{tr}_2}{\longmapsto} (\mathcal{Q}';\psi')$, $\mathsf{obs}(\mathsf{tr}_1) = \mathsf{obs}(\mathsf{tr}_2)$ and $\phi' \sim_s \psi'$. Then, $\mathcal{K}_P \approx_t \mathcal{K}_Q$, if $\mathcal{K}_P \sqsubseteq_t \mathcal{K}_Q$ and $\mathcal{K}_Q \sqsubseteq_t \mathcal{K}_P$.*

Example 3. Consider the configuration $\mathcal{K}_{\mathsf{diff}} = (P(k'_E, k'_M);\phi_0)$ which models the fact that the attacker is now in presence of an other passport that the one that produced ϕ_0. We have that $\mathcal{K}_{\mathsf{same}} \not\sqsubseteq_t \mathcal{K}_{\mathsf{diff}}$, which means that the attacker is able to detect the presence of a passport for which he has partial knowledge of a past session (*i.e.*, ϕ_0). To see this, consider the trace from Example 2. It is possible to produce the same trace starting from $\mathcal{K}_{\mathsf{diff}}$, but the resulting frame is then $\phi' = \phi_0 \uplus \{\mathsf{w}_{c,0} \triangleright n_P; \mathsf{w}_{c,1} \triangleright \mathsf{mac}_{\mathsf{err}}\}$ which does not satisfy the test $\mathsf{w}_{c,1} = \mathsf{nonce}_{\mathsf{err}}$ contrary to the frame produced starting from $\mathcal{K}_{\mathsf{same}}$. This corresponds to a well-known unlinkability attack discovered in [4] on French passports. This attack can be easily fixed by using the same error message in both cases. In such a case, the inclusion holds. This is a non trivial inclusion that can be automatically established by the DeepSec verification tool.

For illustrative purposes, we have only considered here a simple scenario for which configurations under study are actually action-deterministic, *i.e.*, where for any s and α there is at most one s' such that $s \overset{\alpha}{\longmapsto} s'$. In practice, we want to consider more complex scenarios involving several passports and readers, which results in configurations that are *not* action-deterministic: several passports can output on the same channel at the same time. In particular, unlinkability is expressed as an equivalence between processes that are not action-deterministic [4]. When considering unlinkability, we also note that using *diff-equivalence* instead of trace equivalence, as is done in Tamarin and Proverif when checking equivalences for unbounded sessions, systematically leads to false attacks [29]. For such properties, one thus has to resort to verifying trace equivalence in the bounded setting. However, the lack of POR techniques supporting non-action-deterministic processes is a major problem, since equivalence verification tools perform very poorly when the state explosion problem is left untamed.

3 Persistent and sleep sets in a nutshell

We review the key concepts of persistent and sleep sets, based on [28] but slightly reformulated. These general concepts apply to an action-deterministic LTS. We thus assume, in this section, a set of states Q, a set of actions T, and a partial transition function $\delta : Q \times T \to Q$. We write $s \xrightarrow{\alpha} s'$ when $s' = \delta(s,\alpha)$. We say that α is *enabled* in state s if there exists an s' such that $s' = \delta(s,\alpha)$. The set of enabled actions in s is written $E(s)$. A state s is *final* when $E(s) = \emptyset$.

Definition 3. *Independence is the greatest relation $\leftrightarrow \subseteq T \times Q \times T$ that is symmetric, irreflexive and such that, for all $(\alpha, s, \beta) \in \leftrightarrow$ (written $\alpha \leftrightarrow_s \beta$):*

- *if $s \xrightarrow{\alpha} s'$ then $\beta \in E(s)$ iff $\beta \in E(s')$;*
- *if $s \xrightarrow{\alpha} s_1$ and $s \xrightarrow{\beta} s_2$, then $s_1 \xrightarrow{\beta} s'$ and $s_2 \xrightarrow{\alpha} s'$ for some s'.*

Persistent sets. A set $T \subseteq E(s)$ is *persistent in* s if, for all non-empty sequences of actions $s = s_0 \xrightarrow{\alpha_0} s_1 \dots s_n \xrightarrow{\alpha_n} s_{n+1}$ such that $\alpha_i \notin T$ for all $0 \leq i \leq n$, we have that $\alpha_n \leftrightarrow_{s_n} \alpha$ for all $\alpha \in T$. We may note that $E(s)$ is persistent in s. In practice, persistent sets may be computed from *stubborn* sets (see [10]).

In the following, we assume a function $\mathsf{p_{set}} : Q \times T^* \to 2^T$ which associates to any state $s \in Q$ and any sequence w such that $s \xrightarrow{w} s'$ with $E(s') \neq \emptyset$, a non-empty set of actions $\mathsf{p_{set}}(s, w)$ which is persistent in s'.

A trace $s_0 \xrightarrow{\alpha_0} s_1 \dots \xrightarrow{\alpha_n} s_{n+1}$ is *persistent*, written $s_0 \xrightarrow{\alpha_0 \dots \alpha_n}_{\mathsf{p_{set}}} s_{n+1}$, if $\alpha_i \in \mathsf{p_{set}}(s_0, \alpha_0 \dots \alpha_{i-1})$ for all $0 \leq i \leq n$.

Proposition 1. *Let s' be a final state that is reachable from s. We have that s' is also reachable from s through a trace that is persistent.*

Sleep sets. If a persistent set contains two independent actions, then the associated search has redundancies. This has lead to the introduction of sleep sets. This technique relies on an arbitrary ordering $<$ on actions. A sleep set execution is an execution $(s_0, \emptyset) = (s_0, z_0) \xrightarrow{\alpha_0} (s_1, z_1) \dots \xrightarrow{\alpha_n} (s_{n+1}, z_{n+1})$ with states in $Q \times 2^T$ such that $s_0 \xrightarrow{\alpha_0 \dots \alpha_n}_{\mathsf{p_{set}}} s_{n+1}$, and for any $0 \leq i \leq n$ we have $\alpha_i \notin z_i$ and $z_{i+1} = \{\beta \in z_i \mid \alpha_i \leftrightarrow_{s_i} \beta\} \cup \{\beta \in \mathsf{p_{set}}(s_0, \alpha_0 \dots \alpha_{i-1}) \mid \beta < \alpha_i,\ \alpha_i \leftrightarrow_{s_i} \beta\}$.

Proposition 2. *Let s' be a final state that is reachable from s (in the original LTS). We have that s' is also reachable from $(s, \emptyset)$ through a sleep set execution.*

4 Concrete LTS for security protocols

In order to apply the POR techniques of Section 3, we need to reformulate trace equivalence as a reachability property of final states in some LTS.

Given a set of handles $W \subseteq \mathcal{W}$, we define $\mathsf{Conf}(W)$ as the set of alive and *quiescent* configurations with a frame of domain W. An alive configuration $(\mathcal{P}; \phi)$ is quiescent if any $P \in \mathcal{P}$ is of the form $\mathtt{in}(c, x).P'$ or $\mathtt{out}(c, t).P'$ (in other words, no τ action can be triggered from it). We define the set of dead configurations over W as $\mathsf{Conf}_\perp(W) = \{(\perp_j; \phi) \mid \mathrm{dom}(\phi) \subseteq W \text{ and } j \in \mathbb{N}\}$.

We define our *trace equivalence LTS* as follows:

- States are of the form $\langle |\mathbb{A} \approx \mathbb{B}| \rangle$ where $\mathbb{A}, \mathbb{B} \subseteq \mathsf{Conf}(W) \cup \mathsf{Conf}_\perp(W)$ for some $W \subseteq \mathcal{W}$, and at least one configuration in $\mathbb{A} \cup \mathbb{B}$ is alive. The *domain* $\mathrm{dom}(s)$ of such a state is W, and its *age* is $\mathsf{age}(s) = \max(\{0\} \cup \{j + 1 \mid (\perp_j, \phi) \in \mathbb{A} \cup \mathbb{B}\})$.
- Actions are of the form $\mathtt{out}(c, \mathsf{w}_{c,i})$ or $\mathtt{in}(c, M)$ with $c \in \mathcal{C}h$, $i \in \mathbb{N}$, $M \in \mathcal{T}(\mathcal{W})$.
- The transition relation is given by

$$s = \langle |\mathbb{A} \approx \mathbb{B}| \rangle \xrightarrow{\alpha} \langle |\mathbb{A}_a \uplus \mathbb{A}_n \uplus \mathbb{A}_g \approx \mathbb{B}_a \uplus \mathbb{B}_n \uplus \mathbb{B}_g| \rangle$$

  where $\mathbb{A}_g$, $\mathbb{A}_a$, $\mathbb{A}_n$ are given below (and $\mathbb{B}_g$, $\mathbb{B}_n$, and $\mathbb{B}_a$ are defined similarly):

  - $\mathbb{A}_a = \{A' \mid \exists A \in \mathbb{A} \text{ such that } A \xmapsto{\alpha} A'' \xmapsto{\tau^*} A' \not\xmapsto{\gamma}\}$,
  - $\mathbb{A}_n = \{(\perp_{\mathsf{age}(s)}; \phi) \mid (\mathcal{P}; \phi) \in \mathbb{A},\ (\mathcal{P}; \phi) \text{ is alive}, (\mathcal{P}; \phi) \not\xmapsto{\alpha}\}$, and

- $\mathbb{A}_g = \mathbb{A} \cap \mathsf{Conf}_\perp(\mathrm{dom}(s))$.

The transitions gather all alternatives that can perform the same output (resp. input) action. Therefore, even if our protocol allows several alternatives for a given observable action, our resulting trace equivalence LTS is action-deterministic. Configurations that cannot execute such an action become ghosts. A ghost configuration $(\perp_i; \phi)$ is a configuration that cannot evolve anymore; its index i will crucially be used to know what other frames were present when it died (see Example 4).

Given a set of configurations $\mathbb{A}$, we define $\mathbb{A}^{\geq i}$ as the set of all configurations of $\mathbb{A}$ that are still alive at age i. More formally, we have that:

$$\mathbb{A}^{\geq i} = \{(\mathcal{P}, \phi) \in \mathbb{A} \mid (\mathcal{P}, \phi) \text{ is alive or } \mathcal{P} = \perp_j \text{ with } j \geq i\}$$

We write $\phi \sqsubseteq_s \psi$ when $\mathrm{dom}(\phi) \subseteq \mathrm{dom}(\psi)$ and both frames are in static equivalence on their common domain, *i.e.*, $\phi \sim_s \psi|_{\mathrm{dom}(\phi)}$. We lift $\sqsubseteq_s$ to a set of frames (and thus configurations): $\phi \sqsubseteq_s \Psi$ when there exists $\psi \in \Psi$ such that $\phi \sqsubseteq_s \psi$.

Definition 4. *A state $s = \langle |\mathbb{A} \approx \mathbb{B}| \rangle$ is* left-bad *when there exists $(\mathcal{P}; \phi) \in \mathbb{A}$ such that:*

- *either $(\mathcal{P}; \phi)$ is a ghost, i.e., $\mathcal{P} = \perp_j$ for some j, and $\phi \not\sqsubseteq_s \mathbb{B}^{\geq j}$;*
- *or $(\mathcal{P}; \phi)$ is alive and $\phi \not\sqsubseteq_s (\mathbb{B} \cap \mathsf{Conf}(\mathrm{dom}(s)))$.*

The notion of being right-bad *is defined similarly, and we say that a state s is* bad *when it is right-bad or left-bad.*

We will see that trace inequivalence implies the existence of a bad state. Thanks to ghosts, this will directly imply the existence of a final bad state. Fundamentally, ghosts are there to avoid that partial-order reduction makes us miss a bad state by not exploring certain transitions. Of course, practical verification algorithms will never perform explorations past a state that corresponds to a inequivalence witness. Note, however, that detecting such states is only possible thanks to complex constraint solving, which we cannot afford in our symbolic POR algorithms. Hence, one important aspect in our design of ghosts is that they lift well to the "unsolved" symbolic setting.

Example 4. Ghosts are crucial to make sure that progressing in the LTS never kills a witness of inequivalence. For instance, consider the two processes:

$P_u = \mathtt{out}(c, u) + (\mathtt{out}(c, n).\mathtt{out}(d, n'))$ where $u \in \{\mathsf{a}, \mathsf{b}\}$ are two public constants.

Consider $s_0 = \langle |(P_\mathsf{a}; \emptyset) \approx (P_\mathsf{b}; \emptyset)| \rangle$ and $s_0 \xrightarrow{\mathtt{out}(c, \mathsf{w}_{c,0})} s_1 \xrightarrow{\mathtt{out}(d, \mathsf{w}_{d,0})} s_2$ where:

- $s_1 = \langle |\{(0; \{\mathsf{w}_{c,0} \triangleright \mathsf{a}\}), A\} \approx \{(0; \{\mathsf{w}_{c,0} \triangleright \mathsf{b}\}), A\}| \rangle$
- $s_2 = \langle |\{(\perp_0; \{\mathsf{w}_{c,0} \triangleright \mathsf{a}\}), A'\} \approx \{(\perp_0; \{\mathsf{w}_{c,0} \triangleright \mathsf{b}\}), A'\}| \rangle$
- $A = (\mathtt{out}(d, n'); \{\mathsf{w}_{c,0} \triangleright n\})$, and $A' = (0; \{\mathsf{w}_{c,0} \triangleright n, \mathsf{w}_{d,0} \triangleright n'\})$.

Note that s_1 is bad because $\{\mathsf{w}_c^0 \triangleright \mathsf{a}\} \not\sim_s \{\mathsf{w}_c^0 \triangleright \mathsf{b}\}$ and s_2 is bad because the ghost configurations are not statically equivalent either. However, without the ghost configurations, s_2 would not be bad (neither left nor right).

Our first contribution is a result that reduces trace equivalence to reachability of a final bad state in our trace equivalence LTS, on which POR techniques can be applied.

Proposition 3. *Let A_0 and B_0 be two alive configurations of same domain, and $s_0 = \langle |\mathbb{A}_0 \approx \mathbb{B}_0| \rangle$ where $\mathbb{A}_0 = \{A \mid A_0 \stackrel{\tau}{\mapsto}{}^* A \not\stackrel{\gamma}{\mapsto}\}$, and $\mathbb{B}_0 = \{B \mid B_0 \stackrel{\tau}{\mapsto}{}^* B \not\stackrel{\gamma}{\mapsto}\}$. The following conditions are equivalent:*

1. *A_0 is trace included in B_0, i.e., $A_0 \sqsubseteq_t B_0$;*
2. *no left-bad state is reachable from s_0 in the trace-equivalence LTS;*
3. *no left-bad, final state is reachable from s_0 in the trace-equivalence LTS.*

5 POR in symbolic semantics

The POR techniques of Section 3 apply to the LTS of Section 4, but this is not directly usable in practice because our trace equivalence LTS is infinitely branching. Symbolic execution is typically used to circumvent such problems, both in traditional software verification [12] and security protocol analysis [19,20]. In this section, we define a symbolic abstraction of our trace equivalence LTS, and we show how it can be used to effectively apply the persistent and sleep set techniques.

5.1 Symbolic equivalence LTS

As is common in symbolic semantics for security protocols [19,20], we rely on *second-order variables*, which will be instantiated by recipes, and *first-order variables*, which will be instantiated by messages. First-order variables are distinct from standard variables occurring in processes to represent input messages. More precisely, when an input is executed symbolically, the associated variable will be substituted by a first-order variable. As a result, standard variables will only occur bound in symbolic processes, while first-order variables will only occur free. Conversely, only first-order variables will be allowed to occur free in processes, frames, and states.

Second-order and first-order variables will respectively be of the form $\mathsf{X}^{c,i}$ and $\mathsf{x}_\phi^{c,i}$ where $c \in \mathcal{C}h$, $i \in \mathbb{N}$, and ϕ is a symbolic frame, *i.e.*, a frame whose terms may contain first-order variables. Intuitively, $\mathsf{X}^{c,i}$ stands for the recipe used for the i^{th} input on channel c, and $\mathsf{x}_\phi^{c,i}$ will be instantiated by the message resulting from that recipe in the context of the frame ϕ. The use of variables with explicit c, i parameters avoids us to deal with freshness or α-renaming issues when implementing the symbolic analysis. We denote $vars^1(t)$ (resp. $vars^2(t)$) the first-order (resp. second-order) variable occurring in t. Finally, $vars(R)$ is the set of handles that occur in a recipe R. We say that a symbolic frame ϕ is *well-founded* if, whenever $\phi(\mathsf{w}_{c,i}) = t$ and $\mathsf{x}_\psi^{d,j} \in vars^1(t)$, we have that ϕ is a strict extension of ψ meaning that $\phi|_{\mathrm{dom}(\psi)} = \psi$ (denoted $\psi \sqsubseteq \phi$), and $\psi \neq \phi$. This well-foundedness condition will obviously be preserved in symbolic executions: if t is the i^{th} output on channel c, it may only depend on inputs received before that output, *i.e.*, at a time where the frame ψ does not contain $\mathsf{w}_{c,i}$. From now on, we impose that all frames are well-founded, which allows us to define the first-order substitution associated to a second-order substitution.

Definition 5 (λ_θ). *Let θ be a substitution mapping second-order variables to recipes. Its* associated first-order substitution *λ_θ is the unique substitution of (infinite) domain $\{\mathsf{x}_\phi^{c,i} \mid vars(\mathsf{X}^{c,i}\theta) \subseteq \mathrm{dom}(\phi)\}$ such that $\lambda_\theta(\mathsf{x}_\phi^{c,i}) = (\mathsf{X}^{c,i}\theta)(\phi\lambda_\theta)$, which can be defined by induction on the size of frame domains.*

We now define symbolic actions and states, and their concretisations. We take *symbolic actions* of the form $\mathtt{out}(c, \mathsf{w}_{c,i})$ and $\mathtt{in}(c, \mathsf{X}^{c,i}, W)$, where $c \in \mathcal{C}h$, $i \in \mathbb{N}$ and $W \subseteq \mathcal{W}$. Given a substitution θ mapping second-order variables to recipes, we define the θ-concretisations of symbolic actions as follows: $\mathtt{out}(c, \mathsf{w}_{c,i})\theta = \mathtt{out}(c, \mathsf{w}_{c,i})$, and $\mathtt{in}(c, \mathsf{X}^{c,i}, W)\theta = \mathtt{in}(c, R)$ when $\mathsf{X}^{c,i}\theta = R \in \mathcal{T}(W)$. We will use constraints which are conjunctions of equations and disequations over (symbolic) terms, *i.e.*, terms that may contain first-order variables. The empty constraint is written $\top$, and conjunction is written $\wedge$ and considered modulo associativity-commutativity.

A *symbolic state* $S = \langle \mathbb{A} \approx \mathbb{B} \rangle_{\mathcal{C}}^{I}$ is formed from a mapping $I : \mathcal{C}h \to \mathbb{N}$ providing input numbers, a constraint $\mathcal{C}$, and two sets $\mathbb{A}$ and $\mathbb{B}$ of *symbolic configurations*, *i.e.*, configurations that may contain first-order variables. We further require that:

- there is at least one alive configurations in $\mathbb{A} \cup \mathbb{B}$;
- all alive configurations in $\mathbb{A} \cup \mathbb{B}$ share the same frame domain, noted $\mathrm{dom}(S)$;
- any ghost configuration in $\mathbb{A} \cup \mathbb{B}$ should have a domain $W \subseteq \mathrm{dom}(S)$;
- processes in configurations do not contain null processes, and do not feature top-level conditionals, parallel and choice operators.

With this in place, we define the solutions of $S = \langle \mathbb{A} \approx \mathbb{B} \rangle_{\mathcal{C}}^{I}$ as the set $\mathsf{Sol}(S)$ containing all the substitutions θ such that:

- $\mathrm{dom}(\theta) = \{\mathsf{X}^{c,i} \mid i < I(c)\}$;
- for any $u = v$ (resp. $u \neq v$) in $\mathcal{C}$, $u\lambda_\theta =_{\mathsf{E}} v\lambda_\theta$ (resp. $u\lambda_\theta \neq_{\mathsf{E}} v\lambda_\theta$).

Given S and $\theta \in \mathsf{Sol}(S)$, we define its θ-concretisation $S\theta$ as $\langle |\mathbb{A}\lambda_\theta \approx \mathbb{B}\lambda_\theta| \rangle$.

Remark 1. Beyond the differences in formalism, our notion of solution is quite close to ones found, *e.g.*, in [19,20], with one difference: when no $\mathsf{x}_\phi^{c,i}$ variable occurs in S, $\theta(\mathsf{X}^{c,i})$ is completely unconstrained. This means that when an input variable is unused in the input's continuations, our solutions are incorrect wrt. the corresponding recipe. We do not need to worry about this mismatch, though, because we only need a symbolic semantics that covers all concrete executions; it does not need to be sound. In fact, our analysis will never rely on the existence of a solution for a given symbolic state. It will never check that a term is deducible, and will almost ignore (dis)equality constraints, only checking for immediate contradictions among them.

We can now define symbolic transitions, and establish their completeness.

Definition 6. *Consider a symbolic state $S = \langle \mathbb{A} \approx \mathbb{B} \rangle_{\mathcal{C}}^{I}$ and a symbolic action A, the possible transitions $S \overset{A}{\twoheadrightarrow} S'$ are defined by mimicking concrete transitions as follows:*

– *We first execute the action A, gathering all possible resulting configurations into a pre-state $S_A = \langle \mathbb{A}' \approx \mathbb{B}' \rangle_{\mathcal{C}}^{I'}$. To be possible, such a transition has to be of the form $A = \mathtt{in}(c, \mathsf{X}^{c,i}, W)$ with $i = I(c)$, or $A = \mathtt{out}(c, \mathsf{w}_{c,i})$ with $i = \#_c(\mathrm{dom}(S))$. The resulting pre-state S_A is not a valid state because it may contain e.g., top-level conditionals, choice operators. This pre-state also includes ghosts $(\bot_n; \phi)$ of the configurations $(\mathcal{P}; \phi)$ of S that could not perform A, where $n = \mathsf{age}(S)$ as defined in the concrete semantics. We define I' to coincide with I on all channels, except on c where $I'(c) = I(c)+1$ when A is an input on c. When executing $A = \mathtt{in}(c, \mathsf{X}^{c,i}, W)$ in a configuration $(\mathcal{P}; \phi)$ of S that can perform an input on c, we use the term $\mathsf{x}^{c,i}_{\phi|_W}$ to substitute for the input variable.*

– *Then we declare $S \xrightarrow{A} S'$ if S' is a state that can be obtained from S_A by repeatedly performing the following operations, until none applies:*
  - *If a configuration features a top-level conditional, the conditional is replaced by one of its branches, and the constraints are enriched accordingly.*
  - *If a configuration features a top-level choice operator, it is replaced by the two configurations where the choices are made.*

  We also require that S' does not have an immediately contradicting constraint, i.e., a constraint containing an equation and its negation.

A perhaps surprising consequence of our definition is that, if $\mathtt{in}(c, \mathsf{X}^{c,i}, W)$ is enabled in S, then any $\mathtt{in}(c, \mathsf{X}^{c,i}, W')$ is also enabled. Allowing smaller domains is important for checking independencies. We also allow larger domains, possibly even larger than $\mathrm{dom}(S)$, mainly because it simplifies the theory, at no cost in practice.

Example 5. Consider arbitrary terms t, u, and $v \neq v'$, and the symbolic state $S = \langle (\mathcal{P}; \psi) \approx (\mathcal{P}; \psi') \rangle_{\top}^{I}$ where $\mathcal{P} = \mathsf{in}(c, x).\mathsf{if}\ x = t\ \mathsf{then}\ \mathsf{out}(c, \mathsf{ok})\ \mathsf{else}\ 0$,

$$\phi = \{\mathsf{w}_{c,0} \mapsto u\}, \quad \psi = \phi \uplus \{\mathsf{w}_{d,0} \mapsto v\} \text{ and } \psi' = \phi \uplus \{\mathsf{w}_{d,0} \mapsto v'\}.$$

We illustrate how the choice of W affects which transitions are possible from state S with action $A = \mathtt{in}(c, \mathsf{X}^{c,i}, W)$, where $i = I(c)$ is the only value that allows this action to execute, and I' coincides with I except on c for which $I'(c) = I(c) + 1$. If $W = \{\mathsf{w}_{c,0}\}$, then there are two possible transitions:

$$S \xrightarrow{A} \langle (\mathsf{out}(c, \mathsf{ok}); \psi) \approx (\mathsf{out}(c, \mathsf{ok}); \psi') \rangle^{I'}_{\mathsf{x}^{c,i}_{\phi} = t} \qquad S \xrightarrow{A} \langle (0; \psi) \approx (0; \psi') \rangle^{I'}_{\mathsf{x}^{c,i}_{\phi} \neq t}$$

If $W = \{\mathsf{w}_{c,0}, \mathsf{w}_{d,0}\}$, four transitions are possible, notably including

$$S \xrightarrow{A} \langle (\mathsf{out}(c, \mathsf{ok}); \psi) \approx (0; \psi') \rangle^{I'}_{\mathsf{x}^{c,i}_{\psi} = t,\ \mathsf{x}^{c,i}_{\psi'} \neq t}.$$

Indeed, we are considering here an input whose recipe may exploit the different frames of our two configurations. It is a priori possible that the resulting message passes the test $x = t$ only in one configuration.

Remark 2. It may be useful to note that the following property is preserved by symbolic execution, though we do not exploit it: in a configuration $(\mathcal{P}; \phi)$ of a state $\langle \mathbb{A} \approx \mathbb{B} \rangle_{\mathcal{C}}^{I}$, the only first-order variables that appear are of the form $\mathsf{x}^{c,i}_{\psi}$ with $\psi \sqsubseteq \phi$ and $i < I(c)$.

Proposition 4. *Let $S = \langle \mathbb{A} \approx \mathbb{B} \rangle^I_{\mathcal{C}}$ be a symbolic state, $\theta \in \mathsf{Sol}(S)$. Let s' and α be such that $S\theta \xrightarrow{\alpha} s'$. There exists S', A and $\theta' \sqsupseteq \theta$ (i.e. $\theta'|_{\mathrm{dom}(\theta)} = \theta$) such that $S \overset{A}{\twoheadrightarrow} S'$, $\theta' \in \mathsf{Sol}(S')$, $\alpha = A\theta'$, and $s' = S'\theta'$. Moreover, if α is of the form $\mathtt{in}(c, R)$, the proposition holds with $A = \mathtt{in}(c, \mathsf{X}^{c,I(c)}, W)$ for any W such that $vars(R) \subseteq W$.*

5.2 Independence relations

We first define the *enabled symbolic independence* relation, and show that it is a sound abstraction of independence for enabled actions. For that, we assume here a notion of *incompatible* constraints. It can be anything as long as two constraints $\mathcal{C}$ and $\mathcal{C}'$ are only declared incompatible when $\mathcal{C} \wedge \mathcal{C}'$ is unsatisfiable. In practice, we only check for immediate contradictions, *i.e.*, the presence of an equation and its negation. This allows us to easily check $\Leftrightarrow^{ee}$ in the implementation.

Definition 7. *Given a symbolic state S, and two symbolic actions A and B enabled in S, we write $A \Leftrightarrow^{ee}_S B$ when:*
- *A and B are neither two inputs nor two outputs on the same channel;*
- *for any $S \overset{A}{\twoheadrightarrow} S_A$, $S \overset{B}{\twoheadrightarrow} S_B$, we have that $S_A \overset{B}{\twoheadrightarrow} S_{AB}$ and $S_B \overset{A}{\twoheadrightarrow} S_{BA}$ for some symbolic states S_{AB} and S_{BA};*
- *for any $S \overset{A}{\twoheadrightarrow} S_A \overset{B}{\twoheadrightarrow} S_{AB}$, and $S \overset{B}{\twoheadrightarrow} S_B \overset{A}{\twoheadrightarrow} S_{BA}$, we have that S_{AB} and S_{BA} have incompatible constraints, or $S_{AB} = S_{BA}$.*

We now turn to defining a sound abstraction of independence between a concretely disabled and enabled action. Intuitively, $A \Leftrightarrow^{de}_S B$ will guarantee that executing concretisations of B cannot enable new concretisations of A.

Definition 8. *Given a symbolic state S, as well as two symbolic actions A and B, we write $A \Leftrightarrow^{de}_S B$ when B is enabled in S, and*
- *either A is not enabled in S' for any S' such that $S \overset{B}{\twoheadrightarrow} S'$;*
- *or A is enabled in S but A/B are not of the form $\mathtt{in}(c, \mathsf{X}^{c,i}, W)/\mathtt{out}(d, \mathsf{w}_{d,j})$ with $\mathsf{w}_{d,j} \in W$.*

Proposition 5. *Let S be a symbolic state and A and B be two symbolic actions. Let $\theta \in \mathsf{Sol}(S)$, $s = S\theta$ and α (resp. β) be a concretisation of A (resp. B).*
- *If $A \Leftrightarrow^{ee}_S B$, and $\alpha, \beta \in E(s)$, then $\alpha \leftrightarrow_s \beta$.*
- *If $A \Leftrightarrow^{de}_S B$, $\alpha \notin E(s)$ and $\beta \in E(s)$, then $\alpha \leftrightarrow_s \beta$.*

Example 6. Let $\mathcal{P} = \mathtt{in}(c, x).\mathtt{out}(c, x) \mid \mathtt{out}(d, t)$, and $S = \langle (\mathcal{P}; \emptyset) \approx (\mathcal{P}; \emptyset) \rangle^{I_0}_{\top}$ with $I_0(c) = 0$ for any $c \in \mathcal{C}h$. We have $\mathtt{in}(c, \mathsf{X}^{c,0}, \emptyset) \Leftrightarrow^{ee}_S \mathtt{out}(d, \mathsf{w}_{d,0})$: inputs and outputs commute, for inputs whose recipes rely on the currently available (empty) domain. We have $\mathtt{in}(c, \mathsf{X}^{c,0}, \emptyset) \Leftrightarrow^{de}_S \mathtt{out}(d, \mathsf{w}_{d,0})$ (the output does not enable new concretisations for the input) but *not* $\mathtt{in}(c, \mathsf{X}^{c,0}, \{\mathsf{w}_{d,0}\}) \Leftrightarrow^{de}_S \mathtt{out}(d, \mathsf{w}_{d,0})$ (the input is feasible, but performing it after the output would enable new concretisations).

5.3 Persistent set computation

Having defined over-approximations of transitions and dependencies, we now describe how to compute, for a state S, a set of actions $T^+(S)$ that yields a persistent set for any concretisation of S. More precisely, we shall compute stubborn sets (cf. [10]).

Our symbolic LTS is still infinitely branching, due to the absence of constraints on inputs domains W. However, when exploring the LTS, it often suffices to consider inputs with a canonical domain, *i.e.*, the domain of the current state. We formalise this by defining the *enabled cover* of a symbolic state S: $EC(S)$ is the set of all actions that are enabled in S, with the constraint that inputs are of the form $\mathtt{in}(c, \mathsf{X}^{c,i}, \mathrm{dom}(S))$. Proposition 4 already ensures that any concrete action in $E(S\theta)$ can be mapped to a symbolic action in $EC(S)$.

Definition 9. *Let S be a symbolic state, A and B be two symbolic actions such that B is enabled in S. We say that $A \Leftrightarrow_S B$ when* (i) $A \Leftrightarrow^{de}_S B$ *and,* (ii) *if A is enabled in S then $A \Leftrightarrow^{ee}_S B$.*

Given a symbolic state S, we say that a set of actions X is a *symbolic stubborn set for S* when $X \cap EC(S) \neq \emptyset$ and, for any $A \in X$ and any execution

$$S = S_1 \overset{B_1}{\twoheadrightarrow} S_2 \ldots S_n \overset{B_n}{\twoheadrightarrow} S_{n+1} \text{ with } B_i \in EC(S_i) \text{ for all } 1 \leq i \leq n$$

such that $A \not\Leftrightarrow_{S_n} B_n$, there exists $1 \leq i \leq n$ such that $B_i \in X$.

We assume a computable function which associates to any symbolic state S such that $EC(S) \neq \emptyset$ a set $T^+(S)$ that is a symbolic stubborn set for S. Computing $T^+(S)$ is typically achieved as a least fixed point computation, initialising the set with an arbitrary action in $EC(S)$, exploring executions that avoid the current set and adding actions B_n when they are dependent with an action already in the set. In this process all transitions in the enabled cover of S and its successors are considered (unless they are in the current set) without caring for the existence of a solution for the visited states. The computation is carried out with each possible action of $EC(S)$ as its initial set, and a result of minimal cardinality is kept. In the worst case, it will be $EC(S)$ itself.

If done in a depth-first fashion, the computation is (a symbolic approximation of) Godefroid's stubborn set computation through first conflict relations [28]. It is however more efficient to perform the explorations in breadth, since the addition of an action along an exploration can potentially prevent the continuation of another exploration. In any case, the details of how $T^+(S)$ is computed do not matter for correctness.

Example 7. Consider the process $P = \mathsf{in}(c, x).Q \mid \mathsf{in}(d, x).\mathsf{out}(d, t).Q'$ where Q, Q' and t are arbitrary. Consider computing $T^+(S)$ for $S = \langle (P; \emptyset) \approx (P; \emptyset) \rangle^{I_0}_{\top}$, initialising the set with $A_0 = \mathtt{in}(c, \mathsf{X}^{c,0}, \emptyset)$. Since $A_0 \Leftrightarrow_S \mathtt{in}(d, \mathsf{X}^{d,0}, \emptyset)$ we have to explore successors of S by the input on d. There is only one, call it S'. We have $A_0 \Leftrightarrow_{S'} \mathtt{out}(d, \mathsf{w}_{d,0})$, so again we consider the successor S'' by the output action. We have $A_1 = \mathtt{in}(c, \mathsf{X}^{c,0}, \{\mathsf{w}_{d,0}\}) \in EC(S'')$ with $A_1 \not\Leftrightarrow_{S''} A_0$, hence we add A_1 to our set. We repeat the process from S. We have that $A_1 \Leftrightarrow_S \mathtt{in}(d, \mathsf{X}^{d,0}, \emptyset)$, then $A_1 \not\Leftrightarrow_{S'} \mathtt{out}(d, \mathsf{w}_{d,0})$. More precisely, we have

that $A_1 \not\Leftrightarrow^{de}_{S'} \mathtt{out}(d, \mathsf{w}_{d,0})$. Hence we add $A_2 = \mathtt{out}(d, \mathsf{w}_{d,0})$ to our set. Because $A_2 \not\Leftrightarrow^{de}_{S} \mathtt{in}(d, \mathsf{X}^{d,0}, \emptyset) = A_3$, we will also add that action in the next iteration. We thus obtain $T^+(S) = \{A_0, A_1, A_2, A_3\}$, satisfying our specification of T^+. This symbolic stubborn set yields the symbolic persistent set $T^+(S) \cap EC(S) = \{\mathtt{in}(c, \mathsf{X}^{c,0}, \emptyset), \mathtt{in}(d, \mathsf{X}^{d,0}, \emptyset)\}$; in that case, no reduction is possible. However, starting with process $P \mid \mathsf{out}(e, t').P'$ and initialising the set with $A_4 = \mathtt{out}(e, \mathsf{w}_{e,0})$ will often lead to a very good reduction, *i.e.*, a singleton.

Proposition 6. *Let S be a symbolic state such that $EC(S) \neq \emptyset$, and $T = \{A\theta \mid A \in T^+(S)\}$. For any $\theta' \in \mathsf{Sol}(S)$, the set $T \cap E(S\theta')$ is persistent in $S\theta'$.*

Having computed symbolic persistent sets, we now define a persistent set assignment $\mathsf{p_{set}}$ for the concrete LTS. By completeness, we know that, for any concrete execution $s_0 \xrightarrow{\alpha_0} s_1 \dots \xrightarrow{\alpha_{n-1}} s_n$ there exists $S_0 \overset{A_0}{\twoheadrightarrow} S_1 \dots \overset{A_{n-1}}{\twoheadrightarrow} S_n$ and $\theta_0 \sqsubseteq \theta_1 \dots \sqsubseteq \theta_n$ with θ_0 the empty substitution, $\theta_i \in \mathsf{Sol}(S_i)$ and $S_i\theta_i = s_i$ for all $i \in [0; n]$, and $A_i\theta_{i+1} = \alpha_i$ for all $i \in [0; n-1]$. We assume a choice function abs which, to each such concrete execution associates a symbolic abstraction: $\mathsf{abs}(s_0, \alpha_0 \dots \alpha_{n-1}) = (S_0, S_1, \dots, S_n)$. We can assume that the choice is compatible with prefixing:

$\mathsf{abs}(s_0, \alpha_0 \dots \alpha_n) = (S_i)_{0 \leq i \leq n+1}$ implies $\mathsf{abs}(s_0, \alpha_0 \dots \alpha_{n-1}) = (S_i)_{0 \leq i \leq n}$.

Building on this, we define $\mathsf{p_{set}}(s_0, \alpha_0 \dots \alpha_{n-1}) = \{A\theta \mid A \in T^+(S_n)\} \cap E(S_n\theta_n)$ where $\mathsf{abs}(s_0, \alpha_0 \dots \alpha_{n-1}) = (S_0, \dots, S_n)$, which, by Proposition 6, is a persistent set in s_n (uniquely defined as the state reachable from s_0 after $\alpha_0 \dots \alpha_{n-1}$). In other words, we obtain the persistent set for a concrete state from the symbolic persistent set of one of its symbolic abstractions, but we choose this abstraction depending on the concrete execution and not only its resulting state.

With this in place, Proposition 1 guarantees that for any execution from s_0 to a final state s_f, there exists a persistent execution (wrt. $\mathsf{p_{set}}$) from s_0 to s_f. Hence, the search for final bad states can be restricted to only explore concretisations of *symbolic persistent traces*, *i.e.*, symbolic executions where the only transitions considered for a state S are those in $T^+(S) \cap EC(S)$.

5.4 Symbolic sleep sets

We finally describe how we implement sleep sets symbolically. We shall define a symbolic LTS with sleep sets, whose states (S, Z) compound a symbolic state S and a set of symbolic actions Z. The sleep set technique relies on a strict ordering of actions, but the order is only relevant for comparing independent actions, which do not have the same skeleton (the skeleton of an action denotes its input/output nature and its channel). Thus, we assume a strict total order $<$ on action skeletons, and lift it to symbolic and concrete actions. Then, a sleep set execution in our symbolic LTS is any execution

$$(S_0, \emptyset) = (S_0, Z_0) \overset{A_0}{\twoheadrightarrow} (S_1, Z_1) \dots (S_n, Z_n) \overset{A_n}{\twoheadrightarrow} (S_{n+1}, Z_{n+1})$$

such that for $0 \leq i \leq n$, we have that $A_i \in T^+(S_i) \cap EC(S_i)$, $A_i \notin Z_i$, and $Z_{i+1} = \{B \in Z_i \mid B \Leftrightarrow^{ee}_{S_i} A_i\} \cup \{A' \in T^+(S_i) \cap EC(S_i) \mid A' < A_i,\ A' \Leftrightarrow^{ee}_{S_i} A_i\}$.

These symbolic sleep set executions are complete with respect to the sleep set technique applied to our concrete LTS with the $\mathsf{p_{set}}$ function defined above.

Proposition 7. *Let* $(s_0, \emptyset) \xrightarrow{\alpha_0} (s_1, z_1) \dots \xrightarrow{\alpha_{n-1}} (s_n, z_n)$ *be a sleep set execution in our initial LTS. Then, there is* $(S_0, \emptyset) \overset{A_0}{\twoheadrightarrow} (S_1, Z_1) \dots \overset{A_{n-1}}{\twoheadrightarrow} (S_n, Z_n)$ *a sleep set execution in our symbolic LTS, and substitutions* $\emptyset = \theta_0 \sqsubseteq \theta_1 \dots \sqsubseteq \theta_n$ *such that* $s_i = S_i\theta_i$, $\alpha_i = A_i\theta_{i+1}$ *for* $i \in [1; n-1]$, *and* $s_n = S_n\theta_n$.

Example 8. Let S be the state from Example 7. Starting with $(S, \emptyset)$, we may perform two transitions in the sleep LTS: $A_0 = \mathtt{in}(c, \mathsf{X}^{c,0}, \emptyset)$ and $A_3 = \mathtt{in}(d, \mathsf{X}^{d,0}, \emptyset)$. Assuming that $S \overset{A_0}{\twoheadrightarrow} S_c$ and $A_0 > A_3$, we have $(S, \emptyset) \overset{A_0}{\twoheadrightarrow} (S_c, \{A_3\})$. Assuming now that Q starts with another input on c, the persistent set for S_c will contain inputs on c and d. However, executing A_3 is not allowed in $(S_c, \{A_3\})$. Intuitively, while the persistent set technique only looks forward, the sleep set technique also takes into account the past, and indicates here that exploring A_3 is not useful after A_0, since it can equivalently be performed before it.

5.5 Collapsing conditionals

The above techniques allow us, in principle, to compute significantly reduced set of symbolic traces whose concretisations contain a witness of non-equivalence when such a witness exists. However, the algorithm for computing persistent sets is quite inefficient when applied on practical case studies: it relies on explorations of their symbolic LTS, which is highly branching and too large due to conditionals. This is a typical problem of symbolic execution, which manifests itself acutely in our setting, where the branching factor of a state is generally the product of those of its configurations. We circumvent this difficulty by observing that stubborn sets for a state (and its sleep set executions) can be computed by analysing a transformed state where conditionals are pushed down. Our transformation can often completely eliminate conditionals in our case studies, and is key to obtaining acceptable performances.

To justify an elementary step of this transformation, we consider a symbolic state S containing a conditional we would like to simplify: $S = S'[\mathtt{if}\ u = v\ \mathtt{then}\ P\ \mathtt{else}\ Q]$ ($S'[\cdot]$ denotes a state with a hole). We require that P and Q are respectively of the form $\alpha.P'$ and $\beta.Q'$ where α and β have the same skeleton. We make the observation that, independently of the execution and the evaluation of the test $u = v$, the same action will be released and, in case of outputs, the precise output term has little impact in the context of our symbolic analysis. Following this intuition, we would like to postpone the conditional by considering $S^c = S'[\gamma.\mathtt{if}\ u = v\ \mathtt{then}\ P'\ \mathtt{else}\ Q']$, where γ is either the input $\alpha = \beta$, or a well-chosen combination of the outputs α and β. The choice of γ should ensure that the transformation cannot create action independencies that did not hold before the transformation. Formally, we assume a fresh function symbol Δ of arity 4, and take $S^c = S'[T^c]$ where T^c is defined as:

$$\mathsf{in}(c, x).\mathtt{if}\ u = v\ \mathtt{then}\ P'\ \mathtt{else}\ Q' \text{ when } (\alpha, \beta) = (\mathsf{in}(c, x), \mathsf{in}(c, x))$$

$$\mathsf{out}(c, \Delta(t, t', u, v)).\mathtt{if}\ u = v\ \mathtt{then}\ P'\ \mathtt{else}\ Q' \text{ when } (\alpha, \beta) = (\mathsf{out}(c, t), \mathsf{out}(c, t'))$$

Proposition 8. *For any execution $S = S_0 \overset{A_1}{\twoheadrightarrow} S_1 \ldots \overset{A_n}{\twoheadrightarrow} S_n$, there is an execution $S^c = T_0 \overset{A_1}{\twoheadrightarrow} T_1 \ldots \overset{A_n}{\twoheadrightarrow} T_n$, such that, for any A and $i \in [1; n]$, $A \Leftrightarrow_{T_{i-1}} A_i$ (resp. $A_i \Leftrightarrow_{T_{i-1}} A$) implies $A \Leftrightarrow_{S_{i-1}} A_i$ (resp. $A_i \Leftrightarrow_{S_{i-1}} A$).*

Hence, $T^+(S^c)$ is a symbolic stubborn set for S and any sleep set execution from S is also a sleep set execution from S^c.

Repeatedly applying this result, we can eliminate most conditionals from our protocols, and compute stubborn sets and sleep set executions efficiently.

6 Implementation and benchmarks

The results of the previous sections allow us to compute a set of symbolic actions, that can be used to restrict the search when looking for a witness of non-equivalence. By Proposition 3, $(P_1; \emptyset) \not\approx (P_2; \emptyset)$ iff a bad state can be reached from $s_0 = \langle |\mathbb{B}_1 \approx \mathbb{B}_2| \rangle$, where $\mathbb{B}_i = \{\mathcal{K}_i \mid (P_i; \emptyset) \overset{\tau}{\mapsto}{}^* \; \mathcal{K}_i \overset{I}{\not\mapsto}\}$. By Proposition 2, this implies the existence of a sleep set execution in our trace equivalence LTS from $(s_0, \emptyset)$ to a bad state. By Proposition 7, this implies the existence of a concrete execution whose underlying symbolic trace $S_0 = \langle \mathbb{B}_1 \approx \mathbb{B}_2 \rangle^{I_0}_{\top} \overset{A_0}{\twoheadrightarrow} \ldots \overset{A_n}{\twoheadrightarrow} S_{n+1}$ is a sleep set execution in our symbolic LTS. Such symbolic traces can be computed.

6.1 Implementation

To concretely realise and evaluate our techniques, we have implemented our symbolic analysis as a standalone library called Porridge [7], and have interfaced it with Apte in the first place, and then with its successor DeepSec, once this tool has been made available [21]. These tools perform an exhaustive search for non-equivalence witnesses using symbolic execution. Conceptually, this search can be seen as a naive symbolic exploration, combined with an elaborate constraint solving procedure. The two aspects being orthogonal, we can straightforwardly obtain a correct optimisation by restricting the symbolic exploration according to the set of traces computed by Porridge.

Porridge. The library is open-source, written in OCaml. The code implements exactly the techniques presented above, with only a few minor additions.It consists of ∼6k LoC. Performance-wise, we heavily make use of hashconsing and memoization, but not from multicore programming yet. The design of the library, with an independent POR functor, makes it easy to apply symbolic POR analyses to other LTS; we can already perform POR for trace inclusion, and expect to use this flexibility to consider slightly different protocol semantics.

Integration in Apte and DeepSec. As mentioned above, Apte and DeepSec are based on constraint solving procedures on top of which an exhaustive and naive symbolic executions exploration is performed. This exploration is naive in the sense that all interleavings are considered (except for the specific case of action-deterministic protocols already discussed in introduction). We have shown that restricting the exploration to symbolic sleep set traces still yields a

Test	Size	Time (ratio)	Explorations (ratio)	Time (s)
BAC (unlinkability)	4	7.6	7.23	12.23
Private Auth. (anonymity)	2	1.25	2.71	0.04
Private Auth. (anonymity)	3	1.67	4.01	0.04
Private Auth. (anonymity)	4	8.21	10.51	1.17
Private Auth. (anonymity)	5	14.89	16.61	10.57
Private Auth. (anonymity)	6	60.2	36.75	4864
Private Auth. (unlinkability)	2	2.29	9.6	0.16
Private Auth. (unlinkability)	3	14.06	29.77	79.57
Private Auth. (unlinkability)	4	46.2	46.69	7171
Feldhofer (anonymity)	2	1	4.72	0.03
Feldhofer (anonymity)	3	4.63	7.08	0.37
Feldhofer (anonymity)	4	22.47	16.3	544.93
Feldhofer (unlinkability)	4	36.27	22.58	1510.09

Table 1. Relative speed-up and reduction of explorations with Porridge vs. without Porridge. In the last column, we show the computation time without Porridge. The size refers to the total number of processes in parallel.

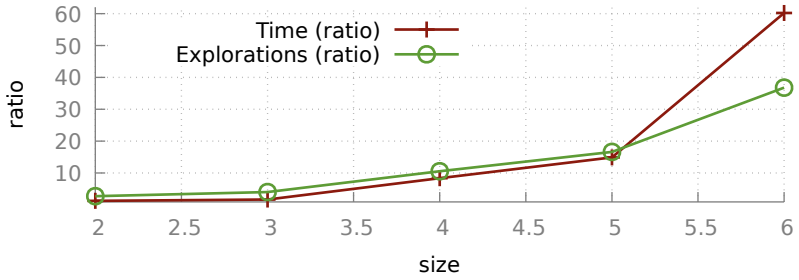

Fig. 2. Relative speed-up and reduction of explorations with Porridge vs. without Porridge on Private Authentication (ANO) of different sizes.

decision procedure for trace equivalence. This restriction is easily implemented, as lightweight modifications (∼500 LoC) of Apte and DeepSec.Note that the differences between the semantics presented in Section 2 and the ones used by those tools can easily be ignored by slightly restricting the class of protocols. Concretely, we exploit the class of protocols with non-blocking outputs as done in [9], which is not restrictive.

6.2 Experimental evaluation

We have carried out numerous benchmarks, focusing on DeepSec since it is both more general and more efficient than Apte, and measuring the improvements brought by Porridge in terms of computation time and number of explorations. The latter is also a good indicator of the effectiveness of the reduction achieved since it represents the number of times DeepSec explores an action and applies its costly constraint solving procedure.

Case studies. We verify some privacy properties on several real-life protocols of various sizes by modifying the number of sessions being analysed. We model unlinkability [29,4] of the BAC protocol [4], of Private Authentication [2] and of Feldhofer [26], and anonymity as well for some of them. The results are shown in Table 1 and make use of processes that are not action-deterministic.

Setup. We run DeepSec and Porridge both compiled with OCaml 4.06.0 on a server running Ubuntu 16.04.5 (Linux 4.4.0) with 12∗2 Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz and 256G of RAM. We run each test on a single core with a time-out of 2 hours (real-time) and maximal memory consumption of 10GB.

Results. We report in Table 1 the relative speed-up of computation time and the reduction of explorations brought by Porridge. We plot the same information for numerous sizes of Private Authentication in Figure 2. We observe that speed-ups are closely related to the reduction achieved on the number of explorations. As the size of protocols increases, Porridge quickly speeds up computations by more that one order of magnitude.

7 Conclusion

We have presented the first POR technique that is applicable to verifying trace equivalence properties of security protocols, without any action-determinism assumption. Our contributions are: an equivalence LTS that recasts trace equivalence as a reachability property; a symbolic abstraction of the equivalence LTS on which persistent and sleep set techniques can be effectively computed; a collapse of conditionals that significantly speeds up these computations. Our technique applies to a wide class of protocols, has been implemented as a library and integrated in the state-of-the-art verifier DeepSec, showing significant performance improvements on case studies.

Compared to (our) earlier work on POR for protocol equivalences [8,9], we follow a radically different approach in this paper to obtain a technique that applies without any action-determinism assumption. In the action-deterministic case, the two techniques achieve similar but incomparable reductions: sleep sets are more efficient on *improper blocks*, but the focused behavior of *compression* is unmatched with sleep sets. Finally, we note that although sleep sets allow to recover a form of *dependency constraint*, we do not know how to justify its use in practice outside of the action-deterministic case. We hope that future work will allow to unify and generalize both techniques.

A crucial aspect of our new approach is that it manages to leverage classic POR techniques, namely persistent and sleep sets, for use in our specific security setting. In fact, we view this work as a first step towards bridging the gap between standard POR and security-specific techniques. As usual in POR, many variations (*e.g.*, in how we integrate with the equivalence verifiers) and approximations (*e.g.*, in independencies or stubborn set computations) should be explored to look for performance gains. The recent work on *dynamic POR* [27,3] (DPOR), which aims to find a trade-off between performance and quality of the computed persistent sets, is of particular interest here, though it is unclear at this

point to which extent generic results can be extracted from the above-mentioned works for re-use in our security setting.

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 C. Fournet. Private authentication. *Theoretical Computer Science*, 322(3):427–476, 2004.
3. P. Abdulla, S. Aronis, B. Jonsson, and K. Sagonas. Optimal dynamic partial order reduction. *ACM SIGPLAN Notices*, 49(1):373–384, 2014.
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. Cuellar, and L. T. Abad. Formal analysis of saml 2.0 web browser single sign-on: Breaking the saml-based single sign-on for google apps. In *Proc. 6th ACM Workshop on Formal Methods in Security Engineering (FMSE 2008)*, pages 1–10, 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'05)*, LNCS. Springer, 2005.
7. D. Baelde, S. Delaune, and L. Hirschi. Porridge, an OCaml library implementing POR techniques for checking trace equivalence of security protocols. `https://hal.inria.fr/hal-01821474`.
8. D. Baelde, S. Delaune, and L. Hirschi. Partial order reduction for security protocols. In *Proceedings of the 26th International Conference on Concurrency Theory (CONCUR'15)*, volume 42 of *LIPIcs*, pages 497–510, Madrid, Spain, 2015. Leibniz-Zentrum für Informatik.
9. D. Baelde, S. Delaune, and L. Hirschi. A Reduced Semantics for Deciding Trace Equivalence. *Logical Methods in Computer Science*, 13(2:8):1–48, 2017.
10. D. Baelde, S. Delaune, and L. Hirschi. POR for Security Protocols Equivalences : Beyond Action-Determinism. Technical report, `https://arxiv.org/abs/1804.03650`, 2018.
11. C. Baier and J.-P. Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.
12. R. Baldoni, E. Coppa, D. C. D'Elia, C. Demetrescu, and I. Finocchi. A survey of symbolic execution techniques. *ACM Computing Surveys*, 51(3), 2018.
13. D. Basin, C. Cremers, and S. Meier. Provably repairing the ISO/IEC 9798 standard for entity authentication. *Journal of Computer Security*, 21(6):817–846, 2013.
14. D. Basin, J. Dreier, and R. Sasse. Automated symbolic proofs of observational equivalence. In *Proc. 22nd ACM Conference on Computer and Communications Security (CCS'15)*, pages 1144–1155. ACM, 2015.
15. B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *Proc. 14th Computer Security Foundations Workshop (CSFW'01)*, pages 82–96. IEEE Computer Society Press, 2001.
16. M. Bruso, K. Chatzikokolakis, and J. den Hartog. Formal verification of privacy for RFID systems. In *Proc. 23rd IEEE Computer Security Foundations Symposium (CSF'10)*. IEEE Computer Society Press, 2010.

17. R. Chadha, c. Ciobâcă, and S. Kremer. Automated verification of equivalence properties of cryptographic protocols. In *Proc. 21st European Symposium on Programming Languages and Systems (ESOP'12)*, LNCS. Springer, 2012.
18. V. Cheval. Apte: an algorithm for proving trace equivalence. In *Proc. 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14)*, 2014.
19. V. Cheval, H. Comon-Lundh, and S. Delaune. Trace equivalence decision: Negative tests and non-determinism. In *Proc. 18th Conference on Computer and Communications Security (CCS'11)*. ACM Press, 2011.
20. V. Cheval, V. Cortier, and S. Delaune. Deciding equivalence-based properties using constraint solving. *Theoretical Computer Science*, 492:1–39, June 2013.
21. V. Cheval, S. Kremer, and I. Rakotonirina. Deepsec: Deciding equivalence properties in security protocols - theory and practice. In *Proc. 39th IEEE Symposium on Security and Privacy (S&P'18)*. IEEE Computer Society Press, 2018.
22. E. Clarke, S. Jha, and W. Marrero. Partial order reductions for security protocol verification. In *Proc. 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 503–518. Springer, 2000.
23. C. J. F. Cremers and S. Mauw. Checking secrecy by means of partial order reduction. In *System Analysis and Modeling*. Springer, 2005.
24. S. Delaune and L. Hirschi. A survey of symbolic methods for establishing equivalence-based properties in cryptographic protocols. *Journal of Logical and Algebraic Methods in Programming*, 2016.
25. 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.
26. M. Feldhofer, S. Dominikus, and J. Wolkerstorfer. Strong authentication for RFID systems using the AES algorithm. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 357–370. Springer, 2004.
27. C. Flanagan and P. Godefroid. Dynamic partial-order reduction for model checking software. In *ACM Sigplan Notices*, volume 40, pages 110–121. ACM, 2005.
28. P. Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems*. PhD thesis, Université de Liège, 1995.
29. L. Hirschi, D. Baelde, and S. Delaune. A method for verifying privacy-type properties: the unbounded case. In *Proc. 37th IEEE Symposium on Security and Privacy (S&P'16)*, pages 564–581, San Jose, California, USA, May 2016.
30. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. 2nd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, volume 1055 of *LNCS*, pages 147–166. Springer-Verlag, 1996.
31. S. Meier, B. Schmidt, C. J. F. Cremers, and D. Basin. The TAMARIN prover for the symbolic analysis of security protocols. In *Proc. 25th International Conference on Computer Aided Verification (CAV'13)*, pages 696–701. Springer, 2013.
32. S. Mödersheim, L. Viganò, and D. Basin. Constraint differentiation: Search-space reduction for the constraint-based analysis of security protocols. *Journal of Computer Security*, 18(4):575–618, 2010.
33. D. Peled. Ten years of partial order reduction. In *Proc. 10th International Conference on Computer Aided Verification (CAV'98)*, LNCS. Springer, 1998.