# Trace Equivalence Decision: Negative Tests and Non-determinism*

Vincent Cheval
LSV, ENS Cachan & CNRS
cheval@lsv.ens-cachan.fr

Hubert Comon-Lundh
LSV, ENS Cachan & CNRS
comon@lsv.ens-cachan.fr

Stéphanie Delaune
LSV, ENS Cachan & CNRS
delaune@lsv.ens-cachan.fr

## ABSTRACT

We consider security properties of cryptographic protocols that can be modeled using the notion of trace equivalence. The notion of equivalence is crucial when specifying privacy-type properties, like anonymity, vote-privacy, and unlinkability.

In this paper, we give a calculus that is close to the applied pi calculus and that allows one to capture most existing protocols that rely on classical cryptographic primitives. First, we propose a symbolic semantics for our calculus relying on constraint systems to represent infinite sets of possible traces, and we reduce the decidability of trace equivalence to deciding a notion of symbolic equivalence between sets of constraint systems. Second, we develop an algorithm allowing us to decide whether two sets of constraint systems are in symbolic equivalence or not. Altogether, this yields the first decidability result of trace equivalence for a general class of processes that may involve else branches and/or private channels (for a bounded number of sessions).

## Categories and Subject Descriptors

D.2.4 [**Program Verification**]: Formal Methods

## General Terms

Security, Verification

## 1. INTRODUCTION

Security protocols are widely used today to secure transaction that rely on public channels like the Internet. It is therefore essential to obtain as much confidence as possible in their correctness. Starting in the 80s, many works have been devoted to the use of formal methods to analyse the security of these protocols (*e.g.* [17, 24]). In the case of a bounded number of sessions, secrecy preservation is co-NP-complete [21, 24], and for an unbounded number of sessions, several decidable classes have been identified (*e.g.* [17, 23]). Many tools have also been developed to automatically verify cryptographic protocols (*e.g.* AVISPA [6], ProVerif [9]).

Until recently, most efforts and successes only concerned trace properties, *i.e.* security properties that can be checked on each individual sequence of messages corresponding to an execution of the protocol. Secrecy and authentication are typical examples of trace properties. There are however several security properties, which cannot be defined as trace properties and require a notion of *behavioural equivalence*. We focus here on the notion of *trace equivalence* which is well-suited for the analysis of security protocols. Intuitively, two processes $P$ and $Q$ are trace equivalent, denoted $P \approx_t Q$, if any experiment performed by an attacker on both processes lead to the emission of two sequences of messages that are indistinguishable, *i.e.* the attacker can not observe any difference between these two sequences. The notion of trace equivalence is weaker than the notion of *observational equivalence* that has been the subject recently of several works [10, 15, 25]. Trace equivalence is probably more adequate to the formalization of privacy-type properties. Originally, observational equivalence is a bisimulation-based equivalence notion that has been introduced as a proof technique for trace equivalence [4]. In the present paper, we are interested in automating the proofs of trace equivalence.

*Related work.* A line of works consists in designing stronger notions of equivalences that imply observational equivalence (and thus trace equivalence). This approach has for instance been used in [7, 8, 25], relying on constraint solving techniques. ProVerif implements an algorithm, based on Horn clauses and dedicated resolution strategies, which is able to establish the observational equivalence between two processes written in the applied pi calculus [10]. However, all these methods check a stronger equivalence than observational equivalence and fail on some simple toy examples. Unfortunately, this is exactly the kind of situations we encountered in several case studies, *e.g.* the private authentication protocol [3], and e-passport protocols [5]. If we restrict our attention to *simple processes* with trivial else branches, then the strong notion of equivalence between two positive constraint systems used in [7] (or [25, 11]), is sufficient to decide trace equivalence. Another procedure for deciding trace equivalence has also been proposed in [18].

Another line of works [20, 12] is based on an extension of the small attack property of [24]: they show that, if two processes are not equivalent, then there must exist a small witness of non-equivalence. A decision of equivalence can be derived by checking every possible small witness. Again this result does not apply to protocols with non-trivial else branches.

In summary, there is no known result that is suitable to protocols such as the private authentication protocol [3], or the e-passport protocol described in [5]: these protocols require a conditional

*This work has been partly supported by the ANR projects AVOTÉ and PROSE, by the grant DIGITEO API from Région Île-de-France, and by the INRIA project SecSI.


*CCS'11,* October 17–21, 2011, Chicago, Illinois, USA.

(with a non-trivial else branch) to be modeled in accurate way. Moreover, the notion of equivalence used by ProVerif or the notion of diff-equivalence used in [8] are too strong to conclude on these case studies. For instance ProVerif yields a false negative in these examples, because the two (equivalent) processes do not have the same control structure.

*Our contributions.* Our main contribution consists in dropping the requirements on the conditionals and on the determinacy in previous algorithms that decide trace equivalence: we provide a new algorithm, that decides the trace equivalence of (possibly non-determinate, possibly with non-trivial else branches) processes, without replication and that use standard primitives, namely signature, hash function, pairing, symmetric and asymmetric encryptions.

We show that the trace equivalence of two processes without replication can be reduced to a notion of symbolic equivalence between sets of initial constraint systems, which we show next how to decide. In our class, we can model in particular conditionals (with non-trivial else branches), private channels, and non-deterministic choice. The private authentication protocol [3] and the various versions of the e-passport protocol [5] fall into our class.

This work is built upon a procedure that has been first described in [11]. However, non trivial conditionals and non-determinism force two main generalizations: we need to consider *sets of constraints* instead of individual constraints and we need also to consider *negative atomic constraints*, for instance disequalities. This yields difficult technical problems, which we will describe, as well as their solutions.

# 2. MODEL

This section introduces our process calculus, by giving its syntax and its semantics. This calculus is close to the original applied pi calculus [2] but we consider a fixed set of cryptographic primitives, namely signatures, pairing, hash function, symmetric and asymmetric encryptions. Participants in a protocol are modeled as processes, and the communication between them is modeled by means of message passing.

## 2.1 Syntax

To describe processes, one starts with an infinite set of *names* $\mathcal{N} = \{a, b, \ldots, sk, k, n, m, \ldots\}$ (which are used to model atomic data), an infinite set of *(first-order) variables* $\mathcal{X}^1 = \{x, y, \ldots\}$, and a set $\mathcal{F}$ of *function symbols* which is split into the set $\mathcal{F}_c$ of *constructors* and the set $\mathcal{F}_d$ of *destructors*. More specifically, we consider:

$$\mathcal{F}_c = \{\mathsf{senc}/2,\ \mathsf{aenc}/2,\ \mathsf{pub}/1,\ \mathsf{sign}/2,\ \mathsf{vk}/1,\ \langle\rangle/2,\ \mathsf{h}/1\}$$
$$\mathcal{F}_d = \{\mathsf{sdec}/2,\ \mathsf{adec}/2,\ \mathsf{check}/2,\ \mathsf{proj}_1/1,\ \mathsf{proj}_2/1\}.$$

This signature contains function symbols to model signature, pairing, hash function, symmetric and asymmetric encryptions.

*Terms* are defined as names, variables, and function symbols applied to other terms. Let $\mathsf{F} \subseteq \mathcal{F}$, $\mathsf{N} \subseteq \mathcal{N}$ and $\mathsf{V} \subseteq \mathcal{X}^1$, the set of terms built from $\mathsf{N}$ and $\mathsf{V}$ by applying function symbols in $\mathsf{F}$ is denoted by $\mathcal{T}(\mathsf{F}, \mathsf{N} \cup \mathsf{V})$. We write $vars^1(u)$ for the set of (first-order) variables occurring in a term $u$. The term $u$ is said to be a *ground* term if $vars^1(u) = \emptyset$. We denote by $st(u)$ the set of subterms of $u$. The *constructor terms*, resp. *ground constructor terms*, are those in $\mathcal{T}(\mathcal{F}_c, \mathcal{N} \cup \mathcal{X}^1)$, resp. in $\mathcal{T}(\mathcal{F}_c, \mathcal{N})$. A ground constructor term is also called a *message*.

We model the properties of our cryptographic primitives by means of a term rewriting system. For instance, the first rule models the fact that the decryption of a ciphertext will return the associated plaintext when the right key is used to perform the decryption.

$$\begin{array}{rcl} \mathsf{sdec}(\mathsf{senc}(x,y),y) & \to & x \\ \mathsf{adec}(\mathsf{aenc}(x,\mathsf{pub}(y)),y) & \to & x \\ \mathsf{check}(\mathsf{sign}(x,y),\mathsf{vk}(y)) & \to & x \end{array} \qquad \begin{array}{rcl} \mathsf{proj}_1(\langle x,y\rangle) & \to & x \\ \mathsf{proj}_2(\langle x,y\rangle) & \to & y \end{array}$$

This term rewriting system is convergent. We denote by $t{\downarrow}$ the normal form of $t$. Moreover, to represent messages, we will only consider valid terms. A term $u$ is *valid*, denoted $\mathsf{valid}(u)$, if $v{\downarrow}$ is a constructor term for any $v \in st(u)$.

We now consider a set $\mathcal{X}^2 = \{X, Y, \ldots\}$ of *second-order variables* and we write $vars^2(\cdot)$ the function that returns the set of second-order variables occurring in its argument. A *recipe* is a term built on $\mathcal{F}_c, \mathcal{F}_d$, a set of *parameters* $\mathcal{AX} = \{ax_1, \ldots, ax_n, \ldots\}$, that can be seen as pointers to the hypotheses (or known messages), and variables in $\mathcal{X}^2$. As in the applied pi calculus, all the function symbols are public, *i.e.* available to the attacker. Moreover, names are excluded from recipes: names that are known to the attacker must be given explicitly as hypotheses. We denote by $\Pi$ the set of recipes, *i.e.* $\Pi = \mathcal{T}(\mathcal{F}, \mathcal{AX} \cup \mathcal{X}^2)$. A *ground recipe* $\zeta$ is a recipe that does not contain variables ($vars^2(\zeta) = \emptyset$) but only parameters. We denote by $param(\zeta)$ the set of parameters that occur in $\zeta$. Intuitively, a ground *recipe* records the attacker's computation. It is used as a witness of how some deduction has been performed.

It is well-known that replication leads to undecidability [19] thus, we consider processes without replication. The grammar of our *plain processes* is as follows:

$$\begin{array}{l} P, Q, R := \mathsf{0} \\ \quad P \mid Q \\ \quad P + Q \\ \quad \mathsf{new}\, x.P \\ \quad \mathtt{if}\ M_1 = M_2\ \mathtt{then}\ P\ \mathtt{else}\ Q \\ \quad \mathsf{in}(M, x).P \\ \quad \mathsf{out}(M, N).P \end{array}$$

where $M_1, M_2, M, N$ are terms, and $x$ is a variable. We denote by $fv(P)$ the free variables of $P$. This notion is extended as expected to multiset of processes.

At a particular point in time, while engaging in one or more sessions of one or more protocols, an attacker may know a sequence of messages $u_1, \ldots, u_n$. As in the applied pi calculus, such a sequence of messages is organized into a (concrete) frame $\phi = \{ax_1 \triangleright u_1, \ldots, ax_n \triangleright u_n\}$ where $u_1, \ldots, u_n$ are ground constructor terms. Its domain $dom(\phi)$ is the set $\{ax_1, \ldots, ax_n\}$ whereas its size $|\phi|$ is the integer $n$. A frame $\phi$ defines a substitution $\{ax \mapsto u \mid ax \in dom(\phi), ax \triangleright u \in \phi\}$.

DEFINITION 1. *A* (concrete) process *is a pair* $(\mathcal{P}; \phi)$ *where:*

- $\phi$ *is a* (concrete) frame*; and*
- $\mathcal{P}$ *is a multiset of* plain processes *(defined above) such that* $fv(\mathcal{P}) = \emptyset$.

*Additionally, we require processes to be* variable distinct*,* i.e. *any variable is at most bound once.*

EXAMPLE 1. *We consider a protocol given in [3] designed for transmitting a secret without revealing its identity to other participants. In this protocol, A is willing to engage in communication with B and wants to reveal its identity to B. However, A does not want to compromise its privacy by revealing its identity or the identity of B more broadly. The protocol works as follows:*

$$\begin{array}{rcl} A \to B & : & \mathsf{aenc}(\langle N_a, \mathsf{pub}(A)\rangle, \mathsf{pub}(B)) \\ B \to A & : & \mathsf{aenc}(\langle N_a, \langle N_b, \mathsf{pub}(B)\rangle\rangle, \mathsf{pub}(A)) \end{array}$$

$$(\{\mathsf{in}(u,x).P\} \uplus \mathcal{P}; \phi) \xrightarrow{\mathsf{in}(\xi,\zeta)} (\{P\{x \to t\}\} \uplus \mathcal{P}; \phi) \quad \text{(IN)}$$
$$\text{if } \zeta, \xi \in \mathcal{T}(\mathcal{F}, dom(\phi)), \zeta\phi\downarrow = t, \xi\phi\downarrow = u\downarrow, \mathsf{valid}(t), \text{ and } \mathsf{valid}(u)$$
$$(\{\mathsf{out}(u,t).Q\} \uplus \mathcal{P}; \phi) \xrightarrow{\mathsf{out}(\xi, ax_{|\phi|+1})} (\{Q\} \uplus \mathcal{P}; \phi \cup \{ax_{|\phi|+1} \triangleright t\downarrow\}) \quad \text{(OUT)}$$
$$\text{if } \xi \in \mathcal{T}(\mathcal{F}, dom(\phi)), \xi\phi\downarrow = u\downarrow, \mathsf{valid}(t), \text{ and } \mathsf{valid}(u)$$

**Figure 1: Semantics**

*First $A$ sends to $B$ a nonce $N_a$ and its public key encrypted with the public key of $B$. If the message is of the expected form then $B$ sends to $A$ the nonce $N_a$, a freshly generated nonce $N_b$ and its public key, all of this being encrypted with the public key of $A$. Otherwise, $B$ sends out a "decoy" message:* $\mathsf{aenc}(N_b, \mathsf{pub}(B))$. *This message should basically look like $B$'s other message from the point of view of an outsider. This is important since the protocol is supposed to protect the identity of the participants.*

*A session of role $B$ played by agent $b$ with $a$ can be modeled by the plain process $B(b,a)$ where $N =$ $\mathsf{adec}(y, skb)$. Note that $B$ is not given the value $ska$ but is directly given the value $\mathsf{pub}(ska)$, that is the public key corresponding to $A$'s private key.*

$$\begin{array}{l} B(b,a) \stackrel{\mathsf{def}}{=} \\ \quad \mathsf{new}\, n_b.\mathsf{in}(c,y). \\ \quad \mathsf{if}\ \mathsf{proj}_2(N) = \mathsf{pub}(ska)\ \mathsf{then} \\ \quad \mathsf{out}(c, \mathsf{aenc}(\langle \mathsf{proj}_1(N), \langle n_b, \mathsf{pub}(skb)\rangle\rangle, \mathsf{pub}(ska))).0 \\ \quad \mathsf{else}\ \mathsf{out}(c, \mathsf{senc}(nb, \mathsf{pub}(skb))).0 \end{array}$$

*Intuitively, this protocol preserves anonymity if an attacker cannot distinguish whether $b$ is willing to talk to $a$ (represented by the process $B(b,a)$) or willing to talk to $a'$ (represented by the process $B(b,a')$), provided $a$, $a'$ and $b$ are honest participants. For illustration purposes, we also consider the process $B'(b,a)$ obtained from $B(b,a)$ by replacing* $\mathsf{else}\ \mathsf{out}(c, \mathsf{senc}(nb, \mathsf{pub}(skb))).0$ *by* $\mathsf{else}\ 0$. *We will see that the "decoy" message plays a crucial role to ensure privacy.*

*A (concrete) process representing a session in which $b$ plays $B$ with $a$ is $(\mathcal{P}_0; \phi_0)$ where $\mathcal{P}_0 = B(b,a)$, and*

$$\begin{array}{l} \phi_0 = \{ax_1 \triangleright c,\ ax_2 \triangleright a,\ ax_3 \triangleright a',\ ax_4 \triangleright b, \\ \qquad ax_5 \triangleright \mathsf{pub}(ska),\ ax_6 \triangleright pub(ska'),\ ax_7 \triangleright \mathsf{pub}(skb)\} \end{array}$$

*The purpose of $\phi_0$ is to disclose the names of the agents and their public keys in order to make them available to the attacker.*

## 2.2 Semantics

The operational semantics of processes is defined by the relation $\xrightarrow{\alpha}$ for which some rules are described in Figure 1. Of course, we have also some rules to deal with conditionals, internal communications, and choice operators. This relation transforms a concrete process into a concrete process. Note that the output of a term $t$ is made "by reference" using the next parameter $ax_{|\phi|+1}$. Moreover, we check the validity of the terms that have to be evaluated during the execution. Note also that, since we authorize arbitrary terms for channels, we have to check whether the channel is known by the attacker or not.

EXAMPLE 2. *Continuing Example 1, we have that:*

$$\begin{array}{rcl} (\mathcal{P}_0; \phi_0) & \stackrel{\mathsf{def}}{=} & (\{B(b,a)\}; \phi_0) \\ & & \xrightarrow{\tau} \xrightarrow{\mathsf{in}(ax_1, \mathsf{aenc}(\langle ax_1, ax_5\rangle, ax_7))} (\{P_1\}; \phi_0) \\ & & \xrightarrow{\mathsf{out}(ax_1, ax_8)} (\{0\}; \phi_0 \cup \{ax_8 \triangleright u\}) \end{array}$$

*where $u = \mathsf{aenc}(\langle c, \langle n_b, \mathsf{pub}(skb)\rangle\rangle, \mathsf{pub}(ska))$ and for some plain process $P_1$.*

*Considering the same sequence of reductions starting from the process $(\{B(b,a')\}; \phi_0)$ yields the process:*

$$(\{0\};\ \phi_0 \cup \{ax_8 \triangleright \mathsf{aenc}(n_b, \mathsf{pub}(skb))\}).$$

*Indeed, in such a situation, the equality test will be false and the "decoy" message will be sent.*

Intuitively, two processes are *equivalent* if they cannot be distinguished by any active attacker. Equivalences can be used to formalize many interesting security properties, in particular privacy-type properties, such as those studied in [16]. However, proofs are difficult because of the universal quantification over all contexts. For this reason, we consider here trace equivalence that intuitively captures the same notion. First, we introduce a notion of intruder's knowledge that has been extensively studied (see *e.g.* [1]).

DEFINITION 2. *Two concrete frames $\phi$ and $\phi'$ are statically equivalent, written $\phi \sim \phi'$, when $dom(\phi) = dom(\phi')$ and when:*

- *for any ground recipe $\zeta$ such that $param(\zeta) \subseteq dom(\phi)$, we have that* $\mathsf{valid}(\zeta\phi)$ *if, and only if,* $\mathsf{valid}(\zeta\phi')$*;*
- *for any ground recipes $\zeta, \zeta'$ such that $param(\{\zeta, \zeta'\}) \subseteq dom(\phi)$, and* $\mathsf{valid}(\zeta\phi)$, $\mathsf{valid}(\zeta\phi')$*, we have that $\zeta\phi\downarrow = \zeta'\phi\downarrow$ if, and only if, $\zeta\phi'\downarrow = \zeta'\phi'\downarrow$.*

EXAMPLE 3. *Consider the two concrete frames introduced in Example 2:*

- $\phi_1 = \phi_0 \cup \{ax_8 \triangleright \mathsf{aenc}(\langle c, \langle n_b, \mathsf{pub}(skb)\rangle\rangle, \mathsf{pub}(ska))\}$*;*
- $\phi_1' = \phi_0 \cup \{ax_8 \triangleright \mathsf{aenc}(n_b, \mathsf{pub}(skb))\}$.

*Actually, they are in static equivalence. This is a non-trivial equivalence. Intuitively, there is no test that allows one to distinguish these two frames since the decryption key $skb$ is not available and the message stored in $ax_8$ can not be reconstructed ($n_b$ is not available to the attacker).*

Let $\mathcal{A}$ be the alphabet of actions for our semantics. For every $w \in \mathcal{A}^*$ the relation $\xrightarrow{w}$ on concrete processes is defined in the usual way. For $s \in (\mathcal{A} \setminus \{\tau\})^*$, the relation $\stackrel{s}{\Rightarrow}$ on concrete processes is defined by: $(\mathcal{P}_A; \phi_A) \stackrel{s}{\Rightarrow} (\mathcal{P}_B; \phi_B)$ if, and only if there exists $w \in \mathcal{A}^*$ such that $(\mathcal{P}_A; \phi_A) \xrightarrow{w} (\mathcal{P}_B; \phi_B)$ and $s$ is obtained by erasing all occurrences of $\tau$.

Let $A = (\mathcal{P}_1; \phi_1)$ be an intermediate process. We define the following set:

$$\mathsf{trace}(A) = \{(s, \phi_2) \mid (\mathcal{P}_1; \phi_1) \stackrel{s}{\Rightarrow} (\mathcal{P}_2; \phi_2) \text{ for some } (\mathcal{P}_2; \phi_2)\}.$$

DEFINITION 3 (TRACE EQUIVALENCE $\approx_t$). *Let $A$ and $B$ be two concrete processes. The processes $A$ and $B$ are* trace equivalent, *denoted by $A \approx_t B$, if for every $(s, \phi) \in \mathsf{trace}(A)$ there exists $(s', \phi') \in \mathsf{trace}(B)$ such that $s = s'$ and $\phi \sim \phi'$ (and conversely).*

EXAMPLE 4. *Continuing Example 1, we have that:*

$$(\{B(b,a)\};\phi_0) \approx_t (\{B(b,a')\};\phi_0).$$

*Again, this is a non-trivial equivalence that illustrates the anonymity property. However, as noticed in [14], the "decoy" message plays an important role. Indeed, considering now the process $B'(b,a)$, we have that:*

$$(\{B'(b,a)\};\phi_0) \not\approx_t (\{B'(b,a')\};\phi_0).$$

*This can be easily shown by considering the sequence of actions $s = \mathsf{in}(ax_1, \mathsf{aenc}(\langle ax_1, ax_5\rangle, ax_7)) \cdot \mathsf{out}(ax_1, ax_8)$. We have that $(s,\phi_1) \in \mathsf{trace}((\{B'(b,a)\};\phi_0))$ (as in Example 2). However, this sequence $s$ does not exist for $(\{B'(b,a')\};\phi_0)$.*

Now, we are able to state our main result:

THEOREM 1. *Let $A$ and $B$ be two concrete processes. The problem whether $A$ and $B$ are trace equivalent is decidable.*

# 3. SYMBOLIC SEMANTICS

In this section, we propose a symbolic semantics for our calculus following *e.g.* [8]. By treating inputs symbolically, our symbolic semantics avoids potentially infinite branching of execution trees due to inputs from the environment. Correctness is maintained by associating with each process a set of constraints on terms. We then define symbolic trace equivalence which is shown to be sound and complete with respect to concrete trace equivalence.

## 3.1 Initial constraint systems

An (initial) constraint system represents the possible executions of a protocol once an interleaving has been fixed.

DEFINITION 4. *An* initial constraint system *is either $\bot$ or a tuple $(\Phi; D; E)$ where:*

1. *$\Phi$ is a sequence of the form $\{ax_1 \triangleright v_1, \ldots, ax_m \triangleright v_m\}$ where $v_1, \ldots, v_m$ are terms;*
2. *$D$ is a sequence of* deducibility constraints *$X_1, i_1 \overset{?}{\vdash} u_1 \ \wedge \ \ldots \ \wedge \ X_n, i_n \overset{?}{\vdash} u_n$ where*
   - *$X_1, \ldots, X_n$ are distinct second-order variables, $u_1, \ldots, u_n$ are terms, and $0 \leq i_1 \leq \ldots \leq i_n \leq m$;*
   - *for every $0 \leq k \leq m$, $vars^1(v_k) \subseteq \bigcup_{i_j<k}\{u_j\}$.*
3. *$E$ is a conjunction of equations and disequations between terms such that $vars^1(E) \subseteq \{u_1, \ldots, u_n\}$.*

Note that each variable $x$ occurs first in a deducibility constraint whose right member is exactly $x$. We also assume that the variables that occur in $E$ have been introduced in a deducibility constraint. This allows us to ensure that once the ground recipes associated to the second order variables are fixed, then the values of the first-order variables are uniquely determined.

Note also that all the variables that occur in such a system are free. The variables $X_i$ represent the recipes that might be used to deduce the right-hand side of the deducibility constraint. The indices indicate the *support* of the variable, *i.e.* which initial segment of the frame can be used. Note that $\Phi$ is not a concrete frame: the terms $v_1, \ldots, v_m$ are not necessarily ground and may contain some destructors.

DEFINITION 5. *A* solution *of an initial constraint system $\mathcal{C} = (\Phi; D; E)$ consists of a substitution $\sigma$ mapping $vars^1(\mathcal{C})$ to ground constructor terms and a substitution $\theta$ mapping $vars^2(\mathcal{C})$ to ground recipes such that:*

- *for every $ax_i \triangleright v_i$ in $\Phi$, we have that $\mathsf{valid}(v_i\sigma)$;*
- *for every $X_i, j \overset{?}{\vdash} u_i$ in $D$, $param(X_i\theta) \subseteq \{ax_1, \ldots, ax_j\}$, $\mathsf{valid}(u_i\sigma)$, and $X_i\theta(\Phi\sigma)\downarrow = u_i\sigma\downarrow$;*
- *for every equation $u \overset{?}{=} v$ in $E$, $\mathsf{valid}(u\sigma)$, $\mathsf{valid}(v\sigma)$, and $u\sigma\downarrow = v\sigma\downarrow$;*
- *for every disequation $u \overset{?}{\neq} v$ in $E$, $\neg\mathsf{valid}(u\sigma)$, or $\neg\mathsf{valid}(v\sigma)$, or $u\sigma\downarrow \neq v\sigma\downarrow$.*

*We denote by $\mathsf{Sol}_{\mathsf{init}}(\mathcal{C})$ the set of solutions of $\mathcal{C}$. By convention, we have that $\mathsf{Sol}_{\mathsf{init}}(\bot) = \emptyset$.*

Let $\mathcal{C}$ be an initial constraint system and $\theta$ be a substitution such that $(\sigma,\theta) \in \mathsf{Sol}_{\mathsf{init}}(\mathcal{C})$ for some $\sigma$. Note that once $\theta$ is fixed, the associated substitution $\sigma$ is uniquely defined.

EXAMPLE 5. *Let $\mathcal{C}_1 = (\Phi_1; D_1; E_1)$ where:*

- *$\Phi_1 = \phi_0 \cup \{ax_8 \triangleright v_8\}$ where $v_8$ is equal to $\mathsf{aenc}(\langle \mathsf{proj}_1(\mathsf{adec}(y, skb)), \langle n_b, \mathsf{pub}(skb)\rangle\rangle, \mathsf{pub}(ska))$*
- *$D_1 = \{X, 7 \overset{?}{\vdash} c;\ Y, 7 \overset{?}{\vdash} y;\ Z, 7 \overset{?}{\vdash} c\}$; and*
- *$E_1 = \{\mathsf{proj}_2(\mathsf{adec}(y, skb)) \overset{?}{=} \mathsf{pub}(ska)\}$*

*$\mathcal{C}_1$ is an initial constraint system. We can check that $(\sigma,\theta) \in \mathsf{Sol}_{\mathsf{init}}(\mathcal{C}_1)$ where:*

- *$\theta = \{X \mapsto ax_1, Y \mapsto \mathsf{aenc}(\langle ax_1, ax_5\rangle, ax_7), Z \mapsto ax_1\}$;*
- *$\sigma = \{y \mapsto \mathsf{aenc}(\langle c, \mathsf{pub}(ska)\rangle, \mathsf{pub}(skb))\}$.*

The *structure* of an initial constraint system $\mathcal{C} = (\Phi; D; E)$ is given by $\{(X,i) \mid X, i \overset{?}{\vdash} u \in D \text{ for some } u\}$, and $dom(\Phi)$. We are now able to define our notion of *symbolic equivalence* between sets of initial constraint systems. This notion will be useful later on to define our notion of *symbolic trace equivalence* (see Section 3.2). The remaining of this paper is entirely devoted to the problem of deciding this notion of symbolic equivalence.

DEFINITION 6. *Let $\Sigma$ and $\Sigma'$ be two sets of initial constraint systems having the same structure. We have that $\Sigma \approx_{\mathsf{init}} \Sigma'$ if for all $\mathcal{C} \in \Sigma$, for all $(\sigma,\theta) \in \mathsf{Sol}_{\mathsf{init}}(\mathcal{C})$, there exists $\mathcal{C}' \in \Sigma'$ and a substitution $\sigma'$ such that $(\sigma',\theta) \in \mathsf{Sol}_{\mathsf{init}}(\mathcal{C}')$ and $\Phi\sigma\downarrow \sim \Phi'\sigma'\downarrow$ where $\mathcal{C} = (\Phi; D; E)$, and $\mathcal{C}' = (\Phi'; D'; E')$ (and conversely).*

## 3.2 Symbolic calculus

From a concrete process $(\mathcal{P};\phi)$ we compute the set of initial constraint systems capturing its possible executions, starting from the symbolic process $(\mathcal{P};\phi;\emptyset;\emptyset)$.

DEFINITION 7. *A* symbolic process *is a tuple $(\mathcal{P}; \Phi; D; E)$:*

- *$\mathcal{C} = (\Phi; D; E)$ is an initial constraint system; and*
- *$\mathcal{P}$ is a multiset of plain processes with $fv(\mathcal{P}) \subseteq vars^1(\mathcal{C})$.*

The semantics of symbolic processes is defined by the relation $\xrightarrow{\alpha_s}_s$ for which some rules are described in Figure 2. This relation transforms a symbolic process into a symbolic process. The aim of this symbolic semantics is to avoid the infinite branching due to the inputs of the environment. This is achieved by keeping variables rather than input terms. The constraint system gives a finite representation of the value that these variables are allowed to take. As in our concrete semantics, we define $\xrightarrow{w}_s$ and $\overset{s}{\Rightarrow}_s$.

$$(\{\mathsf{in}(u,x).P\} \uplus \mathcal{P}; \Phi; D; E) \xrightarrow{\mathsf{in}(X,Y)}_s (\{P\} \uplus \mathcal{P}; \Phi; D'; E) \quad (\text{IN}_s)$$
where $D' = D \wedge X, |\Phi| \overset{?}{\vdash} u \wedge Y, |\Phi| \overset{?}{\vdash} x$ and $Y, X$ are fresh variables

$$(\{\mathsf{out}(u,t).P\} \uplus \mathcal{P}; \Phi; D; E) \xrightarrow{\mathsf{out}(X, ax_{|\Phi|+1})}_s (\{P\} \uplus \mathcal{P}; \Phi \cup \{ax_{|\Phi|+1} \triangleright t\}; D'; E) \quad (\text{OUT}_s)$$
with $D' = D \wedge X, |\Phi| \overset{?}{\vdash} u$ and $X$ is a fresh variable

**Figure 2: Symbolic semantics**

DEFINITION 8. *Let $A = (\mathcal{P}_A; \phi_A)$ and $B = (\mathcal{P}_B; \phi_B)$ be two concrete processes. They are in* symbolic trace equivalence *if for every sequence $\mathsf{tr}_s$ of symbolic actions, we have:*

$$\{(\Phi'_A; D'_A; E'_A) \mid (\mathcal{P}_A; \phi_A; \emptyset; \emptyset) \overset{\mathsf{tr}_s}{\Rightarrow}_s (\mathcal{P}'_A; \Phi'_A; D'_A; E'_A)\}$$
$$\approx_{\mathsf{init}}$$
$$\{(\Phi'_B; D'_B; E'_B) \mid (\mathcal{P}_B; \phi_B; \emptyset; \emptyset) \overset{\mathsf{tr}_s}{\Rightarrow}_s (\mathcal{P}'_B; \Phi'_B; D'_B; E'_B)\}$$

We can show that symbolic trace equivalence exactly captures trace equivalence.

THEOREM 2. *Let $A = (\mathcal{P}_A; \phi_A)$ and $B = (\mathcal{P}_B; \phi_B)$ be two concrete processes. They are in trace equivalence if, and only if, they are in symbolic trace equivalence.*

Now, since the symbolic transition system is finite, we only have to show that symbolic trace equivalence is decidable. However, for processes with non-trivial branchings, the sets of constraints in Definition 8 are not reduced to singletons: we have to consider sets (disjunctions) of contraints.

EXAMPLE 6. *Actually, the system $\mathcal{C}_1$ (see Example 5) is one of the constraint systems obtained by applying our symbolic rules on $(\mathcal{P}_0; \phi_0; \emptyset; \emptyset)$ and considering $\mathsf{tr}_s = \mathsf{in}(X,Y) \cdot \mathsf{out}(Z, ax_8)$. The other one (for the same sequence $\mathsf{tr}_s$) is $\mathcal{C}_2 = (\Phi_2; D_2; E_2)$ where $D_2 = D_1$,*

- $\Phi_2 = \phi_0 \cup \{ax_8 \triangleright \mathsf{aenc}(n_b, \mathsf{pub}(skb))\}$, *and*
- $E_2 = \{\mathsf{proj}_2(\mathsf{adec}(y, skb)) \overset{?}{\neq} \mathsf{pub}(ska)\}$.

*For the same sequence $\mathsf{tr}_s$, similar constraint systems, denoted $\mathcal{C}'_1$ and $\mathcal{C}'_2$, can be derived for the process $(\{B(b,a')\}; \phi_0)$. The occurrences of $ska$ will be replaced by $ska'$.*

*To establish symbolic trace equivalence between the processes $(\{B(b,a)\}; \phi_0)$ and $(\{B(b,a')\}; \phi_0)$, we will need in particular to check that $\{\mathcal{C}_1, \mathcal{C}_2\} \approx_{\mathsf{init}} \{\mathcal{C}'_1, \mathcal{C}'_2\}$.*

# 4. FRAMEWORK

Our procedure for deciding symbolic equivalence between sets of constraint systems requires a slightly different setting from the one introduced in Section 3. Therefore, we adapt and generalize several notions and definitions. In particular, our algorithm (see Section 5) considers constructor terms only. In this section, we show how to get rid of destructor symbols and of some recipes, and we prove that our new notion of symbolic equivalence coincides with the one introduced in Section 3 on sets of initial constraint systems.

## 4.1 Frames

The purpose of a frame is to record the sequence of messages (or terms in a symbolic execution) that have been sent by the participants of the protocol. We extend this notion to record some additional information on attacker's deductions. Typically the element $\mathsf{sdec}(X, \zeta), i \triangleright u$ records that, using a decryption with the recipe $\zeta$, on top of a recipe $X$, allows one to get $u$ (at stage $i$). After recording this information in the frame, we may forbid the attacker to use a decryption on top of $X$, forcing him to use this "direct access" from the frame.

DEFINITION 9 (FRAME). *A (ground)* frame $\Phi$ *is a sequence $\Phi = \zeta_1, i_1 \triangleright u_1, \ldots, \zeta_n, i_n \triangleright u_n$ where:*

- *$\zeta_1, \ldots, \zeta_n$ are (ground) recipes;*
- *$i_1, \ldots, i_n$ are integers; and*
- *$u_1, \ldots, u_n$ are (ground) constructor terms.*

*The* domain *of the frame $\Phi$ is $dom(\Phi) = \mathcal{AX} \cap \{\zeta_1, \ldots, \zeta_n\}$. It must be equal to $\{ax_1, \ldots, ax_m\}$ for some $m$ that is called the* size *of $\Phi$. Such a frame $\Phi$ defines a substitution on $dom(\Phi)$.*

In order to restrict the set of recipes we have to work with, we define the following set $\Pi_{\mathsf{r}}$:

$$\Pi_{\mathsf{r}} = \left\{ \xi \in \Pi \;\middle|\; \begin{array}{c} \mathsf{g}(\xi_1, \ldots, \xi_n) \in st(\xi) \text{ for some } \mathsf{g} \in \mathcal{F}_d \\ \Rightarrow \mathsf{Top}(\xi_1) \notin \mathcal{F}_c \end{array} \right\}$$

where $\mathsf{Top}(u)$ denotes the root symbol of $u$.

For instance, $\mathsf{sdec}(\mathsf{senc}(ax_1, ax_1), ax_2)$ is *not* in $\Pi_{\mathsf{r}}$. When checking static equivalence (resp. symbolic equivalence) between frames (constraint systems) that only contain constructor terms, we can restrict ourselves to consider only recipes that are in $\Pi_{\mathsf{r}}$. Thus, in the remainder, we will only consider recipes in $\Pi_{\mathsf{r}}$.

## 4.2 Constraint systems

We slightly generalize the constraint systems introduced in Section 3. Let us explain how and why. According to Section 3, we need to decide symbolic equivalence of sets (disjunctions) of constraint systems, *e.g.* $\{\mathcal{C}_1, \mathcal{C}_2\} \overset{?}{\approx} \{\mathcal{C}'_1, \mathcal{C}'_2\}$. We cannot split these sets and consider instead sets of pairs, because the solutions of, say, $\mathcal{C}_1$ might be covered by both $\mathcal{C}'_1$ and $\mathcal{C}'_2$ and, conversely, some solutions of $\mathcal{C}'_1$ might correspond to solutions of $\mathcal{C}_2$. Now, if we wish to apply a transformation rule, to one of the component, say $\mathcal{C}_1$, our choice must be consistent with the transformation performed on the other components $\mathcal{C}_2, \mathcal{C}'_1, \mathcal{C}'_2$. For instance if we guess that a key is deducible in $\mathcal{C}_1$ using a recipe $\xi$, we must consistently use $\xi$ in the constraints $\mathcal{C}'_1, \mathcal{C}'_2$, and in turn in $\mathcal{C}_2$. In summary, we need to make this choice for the whole pair of sets. Now, if the key is assumed to be non-deducible in $\mathcal{C}_1$, this must also be recorded in the other components. In short, we need to split the solutions into disjoint sets *for every component*. This yields negative constraints, typically a constraint that states that a key is not deducible.

In summary, we extend the constraints, adding some negative information.

DEFINITION 10 (CONSTRAINT SYSTEM). *It is either $\bot$ or a tuple $(\Phi; D; E; E_r; ND)$ where:*

1. $\Phi$ *is a frame, whose size is some* $m$*;*

2. $D$ *is a sequence* $X_1, i_1 \overset{?}{\vdash} u_1;\ \ldots;\ X_n, i_n \overset{?}{\vdash} u_n$ *where*
   - $X_1, \ldots, X_n$ *are distinct second-order variables,* $u_1, \ldots, u_n$ *are constructor terms, and we have that* $0 \leq i_1 \leq \ldots \leq i_n \leq m$*.*
   - *for every* $\xi, i \triangleright u$ *in* $\Phi$*,* $vars^1(u) \subseteq \bigcup_{i_j < i} vars^1(u_j)$*;*

3. $E = \bigwedge_k u_k \overset{?}{=} v_k \wedge \bigwedge_i \forall \tilde{x}_i \cdot [\bigvee_j u_{i,j} \overset{?}{\neq} v_{i,j}]$ *where the terms* $u_k, v_k, u_{i,j}$ *and* $v_{i,j}$ *are constructor terms.*

4. $E_r = \bigwedge_i \zeta_i \overset{?}{=} \zeta'_i \wedge \bigwedge_j \xi_j \overset{?}{\neq} \xi'_j \wedge \bigwedge_k \mathsf{Top}(\beta_k) \overset{?}{\neq} \mathsf{f}_k$ *where* $\zeta_i, \zeta'_i$*,* $\xi_j, \xi'_j$*,* $\beta_k$ *are recipes in* $\Pi_\mathsf{r}$ *and* $\mathsf{f}_k$ *are constructor symbols.*

5. $ND = \bigwedge_i \forall \tilde{x}_i.[u_i \overset{?}{\neq} v_i \vee \bigvee_j k_{i,j} \overset{?}{\not\vdash} w_{i,j}]$ *where* $u_i, v_i, w_{i,j}$ *are constructor terms and* $0 \leq k_{i,j} \leq m$*.*

In Section 2, we define $vars^1(\mathcal{C})$ as the set of all first order variables that occur in $\mathcal{C}$. With this extended definition of a constraint system, the set $vars^1(\mathcal{C})$ will denote the free variables that occur in $\mathcal{C}$, *i.e.* those that do not occur explicitly under a forall quantification.

In order to define the notion of solution for such a constraint system, we have first to give the semantics of the formulas $ND$ and $E$. The formulas $ND$ and $E$ are logic formulas built upon elementary formulas using classical connectives. The semantics for the elementary formulas are given below and is extended as expected to general formulas. Let $\theta$ be a substitution mapping $vars^2(\mathcal{C})$ to ground recipes, and $\sigma$ be a substitution mapping $vars^1(\mathcal{C})$ to ground constructor terms. We have that:

- $\sigma \vDash (i \overset{?}{\not\vdash} u)$ iff $\xi\Phi\sigma\downarrow \neq u\sigma\downarrow$ for any ground recipe $\xi \in \Pi_\mathsf{r}$ with $param(\xi) \subseteq \{ax_1, \ldots, ax_i\}$;
- $\sigma \vDash u \overset{?}{\neq} v$, iff $u\sigma \neq v\sigma$;
- $\theta \vDash \xi_1 \overset{?}{=} \xi_2$ (resp. $\theta \vDash \xi_1 \overset{?}{\neq} \xi_2$) iff $\xi_1\theta = \xi_2\theta$ (resp. $\xi_1\theta \neq \xi_2\theta$);
- $\theta \vDash \mathsf{Top}(\xi) \overset{?}{\neq} \mathsf{f}$ iff $\mathsf{Top}(\xi\theta) \neq \mathsf{f}$.

DEFINITION 11. *A* solution *of* $\mathcal{C} = (\Phi; D; E; E_r; ND)$ *consists of a substitution* $\sigma$ *mapping* $vars^1(\mathcal{C})$ *to ground constructor terms and a substitution* $\theta$ *mapping* $vars^2(\mathcal{C})$ *to ground recipes in* $\Pi_\mathsf{r}$*, such that:*

1. *for every* $X_i, j \overset{?}{\vdash} u_i$ *in* $D$*, we have that* $X_i\theta(\Phi\sigma)\downarrow = u_i\sigma\downarrow$ *and* $param(X_i\theta) \subseteq \{ax_1, \ldots, ax_j\}$*;*
2. $\sigma \vDash ND \wedge E$ *and* $\theta \vDash E_r$*.*

*We denote by* $\mathsf{Sol}(\mathcal{C})$ *the set of solutions of* $\mathcal{C}$*. By convention,* $\mathsf{Sol}(\bot) = \emptyset$*.*

The *structure* of a system $\mathcal{C} = (\Phi; D; E; E_r; ND)$ is given by $E_r, \{(X, i) \mid X, i \overset{?}{\vdash} u \in D\}$, and $\{(\xi, i) \mid \xi, i \triangleright u \in \Phi\}$. Two constraint systems $\mathcal{C}$ and $\mathcal{C}'$ have the *same structure* if their underlying structure are equal.

For this generalized notion of constraint systems, we can define the notion of symbolic equivalence accordingly.

DEFINITION 12. *Let* $\Sigma$ *and* $\Sigma'$ *be two sets of constraint systems having the same structure. We have that* $\Sigma \approx_s \Sigma'$ *if for all* $\mathcal{C} \in \Sigma$*, for all* $(\sigma, \theta) \in \mathsf{Sol}(\mathcal{C})$*, there exists* $\mathcal{C}' \in \Sigma'$ *and a substitution* $\sigma'$ *such that* $(\sigma', \theta) \in \mathsf{Sol}(\mathcal{C}')$ *and* $\Phi\sigma\downarrow \sim \Phi'\sigma'\downarrow$ *where* $\mathcal{C} = (\Phi; D; E; E_r; ND)$*, and* $\mathcal{C}' = (\Phi'; D'; E'; E'_r; ND')$ *(and conversely).*

Note also that given an initial constraint system $\mathcal{C}_0 = (\Phi_0; D_0; E_0)$ that may contain some destructors, we can transform it into an "equivalent" constraint system $\mathcal{C} = (\Phi; D; E; E_r; ND)$ (without destructors) in the sense that:

- for every $(\sigma_0, \theta) \in \mathsf{Sol}_{\mathsf{init}}(\mathcal{C}_0)$ with recipes in $\Pi_\mathsf{r}$, there exists $(\sigma, \theta) \in \mathsf{Sol}(\mathcal{C})$ such that $\Phi_0\sigma_0\downarrow = \Phi\sigma\downarrow$; and
- for every $(\sigma, \theta) \in \mathsf{Sol}(\mathcal{C})$, there exists $(\sigma_0, \theta) \in \mathsf{Sol}_{\mathsf{init}}(\mathcal{C}_0)$ such that $\Phi\sigma\downarrow = \Phi_0\sigma_0\downarrow$.

Roughly, the transformation consists in guessing the possible reductions, in the spirit of [13]. We write $\mathbf{Tr}(\mathcal{C}_0)$ to denote the result of applying the transformation to $\mathcal{C}_0$.

EXAMPLE 7. *Going back to the initial constraint system* $\mathcal{C}_1$ *(resp.* $\mathcal{C}_2$*) described in Example 5 (resp. Example 6), the constraint system* $\mathbf{Tr}(\mathcal{C}_1)$ *is obtained from* $\mathcal{C}_1$ *by applying the substitution* $\{y \mapsto \mathsf{aenc}(\langle y', \mathsf{pub}(ska)\rangle, pub(skb))\}$ *(and normalizing the result) whereas* $\mathbf{Tr}(\mathcal{C}_2)$ *is obtained from* $\mathcal{C}_2$ *by replacing its disequation with* $\forall y' \cdot y \overset{?}{\neq} \mathsf{aenc}(\langle y', \mathsf{pub}(ska)\rangle, \mathsf{pub}(skb))$*. Note that in both cases, we get rid of all the destructors.*

PROPOSITION 1. *Let* $\Sigma = \{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$*,* $\Sigma' = \{\mathcal{C}'_1, \ldots, \mathcal{C}'_\ell\}$ *be two finite sets of initial constraint systems. We have that:*

$$\Sigma \approx_{\mathsf{init}} \Sigma' \quad \Leftrightarrow \quad \{\mathbf{Tr}(\mathcal{C}_i) \mid \mathcal{C}_i \in \Sigma\} \approx_s \{\mathbf{Tr}(\mathcal{C}'_i) \mid \mathcal{C}'_i \in \Sigma'\}$$

# 5. ALGORITHM

The main result of this section is a decision procedure for symbolic equivalence between sets of constraint systems obtained by applying our transformation $\mathbf{Tr}()$ on sets of initial constraint systems.

THEOREM 3. *Given two sets* $\Sigma$*,* $\Sigma'$ *of initial constraint systems, it is decidable whether* $\{\mathbf{Tr}(\mathcal{C}) \mid \mathcal{C} \in \Sigma\} \approx_s \{\mathbf{Tr}(\mathcal{C}') \mid \mathcal{C}' \in \Sigma'\}$*.*

Our decision algorithm works by rewriting pairs of sets of constraint systems, until a trivial failure or a trivial success is found. These rules are branching: they rewrite a pair of sets of constraint systems into two pairs of sets of constraint systems. Transforming the pairs of constraints therefore builds a binary tree. Termination requires to follow a particular strategy that is explained in Section 5.3. The transformation rules are *sound*: if all leaves are success leaves, then the original pair of sets of constraint systems is equivalent. They are finally *complete*: if the two original sets of constraint systems are equivalent then all the leaves are success leaves.

## 5.1 Transformation rules

The transformation rules are displayed in Figure 3 for a single constraint system (we only write the components of the constraint systems that are modified during an application of an instance of a rule). Since we intend to decide symbolic equivalence between two sets of constraint systems, we will then explain how to apply these transformation rules in such a setting (see Section 5.2).

The CONS rule simply guesses whether the top symbol of the recipe is a constructor $\mathsf{f}$. Either it is, and then we can split the

**Transformation rules for satisfiability:**

$$\underline{\text{CONS}(X,\mathsf{f})} : X, i \overset{?}{\vdash} t; E; E_r \quad \begin{cases} \rightarrow & X_1, i \overset{?}{\vdash} x_1; \cdots; X_n, i \overset{?}{\vdash} x_n; E \wedge t \overset{?}{=} \mathsf{f}(x_1,\ldots,x_n); E_r \wedge X \overset{?}{=} \mathsf{f}(X_1,\ldots,X_n) \\ \rightarrow & X, i \overset{?}{\vdash} t\ ;\ E;\ E_r \wedge \mathsf{Top}(X) \neq \mathsf{f} \end{cases}$$

where $x_1,\ldots,x_n, X_1,\ldots X_n$ are fresh variables.

$$\underline{\text{AXIOM}(X,\xi)} :\ X, i \overset{?}{\vdash} u;\ E;\ E_r \quad \begin{cases} \rightarrow & E \wedge u \overset{?}{=} v;\ E_r \wedge X \overset{?}{=} \xi \\ \rightarrow & X, i \overset{?}{\vdash} u;\ E;\ E_r \wedge X \overset{?}{\neq} \xi \end{cases}$$

If $(\xi, j \triangleright v) \in \Phi$ and $i \geq j$.

$$\underline{\text{DEST}(\xi, l \to r, i)} : E; ND \quad \begin{cases} \rightarrow & \mathsf{g}(\xi, X_2,\ldots,X_n), i \triangleright w; X_2, i \overset{?}{\vdash} u_2; \ldots; X_n, i \overset{?}{\vdash} u_n; E_p \wedge v \overset{?}{=} u_1; ND \\ \rightarrow & E; ND \wedge \forall \tilde{x} \cdot [v \neq u_1 \vee i \overset{?}{\not\vdash} u_2 \vee \ldots \vee i \overset{?}{\not\vdash} u_n] \end{cases}$$

If $(\xi, j \triangleright v) \in \Phi$ with $j \leq i$. $X_2,\ldots,X_n$ are fresh variables, and $\tilde{x}$ are the variables that occur in $\mathsf{g}(u_1,\ldots,u_n) \to w$ a fresh renaming of the rewriting rule $l \to r$.

**Additional transformation rules for static equivalence:**

$$\underline{\text{EQ-LEFT-LEFT}(\xi_1,\xi_2)} : E \quad \begin{cases} \rightarrow & E \wedge u_1 \overset{?}{=} u_2 \\ \rightarrow & E \wedge u_1 \overset{?}{\neq} u_2 \end{cases}$$

If $(\xi_1, i_1 \triangleright u_1),\ (\xi_2 \triangleright u_2) \in \Phi$ for some $u_1, u_2, i_1, i_2$.

$$\underline{\text{EQ-LEFT-RIGHT}(\xi_1, X_2)} : E \quad \begin{cases} \rightarrow & E \wedge u_1 \overset{?}{=} u_2 \\ \rightarrow & E \wedge u_1 \overset{?}{\neq} u_2 \end{cases}$$

If $(\xi_1, i_1 \triangleright u_1) \in \Phi$ and $(X_2, i_2 \overset{?}{\vdash} u_2) \in D$ for some $u_1, u_2, i_1, i_2$ such that $i_2 < i_1$.

$$\underline{\text{EQ-RIGHT-RIGHT}(X,\xi)} : X, i \overset{?}{\vdash} u;\ E;\ E_r \quad \begin{cases} \rightarrow & E \wedge u \overset{?}{=} v;\ E_r \wedge X \overset{?}{=} \xi \\ \rightarrow & X, i \overset{?}{\vdash} u;\ E \wedge u \overset{?}{\neq} v;\ E_r \end{cases}$$

where $\xi \in \mathcal{T}(\mathcal{F}_c, dom(\alpha))$ and $v = \xi\alpha$ with $\alpha = \{Y \mapsto u \mid (Y, j \overset{?}{\vdash} u) \in D \wedge j \leq i\}$.

$$\underline{\text{DED-ST}(\xi,\mathsf{f})} : E; ND \quad \begin{cases} \rightarrow & X_1, m \overset{?}{\vdash} x_1;\ \cdots;\ X_n, m \overset{?}{\vdash} x_n;\ E \wedge u \overset{?}{=} \mathsf{f}(x_1,\ldots,x_n);\ ND \\ \rightarrow & E;\ ND \wedge \forall \tilde{x} \cdot [u \neq \mathsf{f}(x_1,\ldots,x_n) \vee m \overset{?}{\not\vdash} x_1 \vee \ldots \vee m \overset{?}{\not\vdash} x_n] \end{cases}$$

If $(\xi, i \triangleright u) \in \Phi$. The sequences $\tilde{x} \overset{\mathsf{def}}{=} x_1,\ldots,x_n$, and $X_1,\ldots,X_n$ are sequences of fresh variables and $m$ denotes the size of $\Phi$.

---

*All rules assume that the equations have a mgu that is eagerly applied to the resulting constraint, that the disequations have been simplified. Moreover, if there exists a constraint $X, i \overset{?}{\vdash} u$ with $u \notin \mathcal{X}^1$ and on which the rule* CONS *and* AXIOM *cannot be applied on it, and the rule* DEST *can not be applied anymore, then we replace $\mathcal{C}$ with $\bot$.*

**Figure 3: Transformation rules**

deducibility constraint, or it is not and we add a disequation on recipes forbidding this. The rule AXIOM also guesses whether a trivial recipe (a left member of the frame, typically an axiom $ax_i$) can be applied. If so, the constraint can simply be removed. Otherwise, we also add a disequation on recipes forbidding it. The DEST rule is more tricky. If $v$ is a term of the frame, that can be unified with a non variable subterm of a left side of a rewrite rule (for instance $v$ is a ciphertext), we guess whether the rule can be applied to $v$. This corresponds to the equation $u_1 \stackrel{?}{=} v$, that yields an instance of $w$, the right member of the rewrite rule, provided that the rest of the left member is also deducible: in case of symmetric encryption, we get a constraint $X_2, j \stackrel{?}{\vdash} u_2$. The various equality rules guess equalities between right-hand sides of deducibility constraints and/or members of the frame. Finally, the last transformation rule is the only rule that is needed to get in addition a static equivalence decision algorithm, as in [1]. Thanks to this rule, if a subterm of the frame is deducible, then there will be a branch in which it is deduced.

The idea behind these rules is to transform a system into simpler ones. Typically, as it is done in [11], we want to consider systems in which right-hand sides of deducibility constraints are distinct variables (assuming that the mgu corresponding to the equations has been applied on the constraints). However, in presence of disequations, putting the systems in such a form does not guarantee anymore that the two resulting systems will be in symbolic equivalence. Let us illustrate this using a simple example.

EXAMPLE 8. *Consider the constraint systems*

- $\mathcal{C} = (ax_1, 1 \triangleright a;\ X, 1 \stackrel{?}{\vdash} x;\ \emptyset;\ \emptyset;\ \emptyset)$*, and*
- $\mathcal{C}' = (ax_1, 1 \triangleright a;\ X, 1 \stackrel{?}{\vdash} x;\ x \stackrel{?}{\neq} \langle a, a\rangle;\ \emptyset;\ \emptyset)$.

*Although these two systems have the expected form, they are* not *in symbolic equivalence (consider for instance the recipe* $\langle ax_1, ax_1\rangle$*).*

Once the system is put in this kind of "pre-solved form", the basic idea will be to continue to apply our transformation rules to "match" the disequations of each constraint system. For this, we want to transform the disequations in which some names or universally quantified variable occur until obtaining disequations that only contain free variables and public function symbols. This will guarantee that there exists a recipe associated to this term and this gives us the way to match it in another constraint system. Once the system is transformed into such a new kind of "solved form", we can now easily conclude. Indeed, since we also take care of static equivalence on the resulting frames, disequations that correspond to public disequality tests are easily transferable from one constraint system to another without any additional checks.

EXAMPLE 9. *Continuing Example 8 and assuming that the pairing operator is the only constructor symbol, we will go on, applying* CONS. *Let* $\Phi_0 = \{ax_1, 1 \triangleright a\}$, $D = \{X_1, 1 \stackrel{?}{\vdash} x_1; X_2, 1 \stackrel{?}{\vdash} x_2\}$, *and* $E_r = \{X \stackrel{?}{=} \langle X_1, X_2\rangle\}$. *One of the resulting pair will be the pair* $(\mathcal{C}_1, \mathcal{C}'_1)$ *where:*

- $\mathcal{C}_1 = (\Phi_0; D; x \stackrel{?}{=} \langle x_1, x_2\rangle; E_r; \emptyset)$*;*
- $\mathcal{C}'_1 = (\Phi_0; D; x \stackrel{?}{=} \langle x_1, x_2\rangle \wedge [x_1 \stackrel{?}{\neq} a \vee x_2 \stackrel{?}{\neq} a]; E_r; \emptyset)$*;*

*Now, by applying the* AXIOM *rule twice, one of the resulting pair will be the pair* $(\mathcal{C}_2; \mathcal{C}'_2)$ *where:*

- $\mathcal{C}_2 = (\Phi_0; \emptyset; x \stackrel{?}{=} \langle a, a\rangle; X = \langle ax_1, ax_1\rangle; \emptyset)$*; and*
- $\mathcal{C}'_2 = \bot$ *since the disequations will be trivially not satisfied.*

*These two constraint systems are not in symbolic equivalence.*

Now, to ensure that we will reach such a solved form in which all the disequations are matched, the rule EQ-RIGHT-RIGHT plays an important role.

EXAMPLE 10. *Consider the two constraint systems:*

- $\mathcal{C} = (\Phi_0; D_0; x \stackrel{?}{\neq} \mathsf{h}(y) \wedge x \stackrel{?}{\neq} y; \emptyset; \emptyset)$*;*
- $\mathcal{C}' = (\Phi_0; D_0; \emptyset; \emptyset; \emptyset)$

*where* $\Phi_0 = \{ax_1, 1 \triangleright a\}$ *and* $D_0 = \{X, 1 \stackrel{?}{\vdash} x;\ Y, 1 \stackrel{?}{\vdash} y\}$.

*We could apply* CONS *replacing* $x$ *with* $\mathsf{h}(x')$ *to simplify the disequation* $x \stackrel{?}{\neq} \mathsf{h}(y)$ *into* $x' \stackrel{?}{\neq} y$. *However, this operation will transform the other disequation, namely* $x \stackrel{?}{\neq} y$, *into* $\mathsf{h}(x') \stackrel{?}{\neq} y$. *Consequently, one of the resulting pair would be made up of two systems on which the* CONS *rule is again applicable. Instead of this, since the disequation* $x \stackrel{?}{\neq} \mathsf{h}(y)$ *does not contain any name, it can be matched to the other system, so we apply* EQ-RIGHT-RIGHT. *This leads us to the pairs* $(\bot, \mathcal{C}'_1)$ *and* $(\mathcal{C}; \mathcal{C}'_2)$ *where:*

- $\mathcal{C}'_1 = (\Phi_0; \{Y, 1 \stackrel{?}{\vdash} y\}; x \stackrel{?}{=} \mathsf{h}(y); X \stackrel{?}{=} \mathsf{h}(Y); \emptyset)$*; and*
- $\mathcal{C}'_2 = (\Phi_0; D_0; x \stackrel{?}{\neq} \mathsf{h}(y); \emptyset; \emptyset)$.

*From the pair* $(\bot, \mathcal{C}'_1)$ *we will conclude that symbolic equivalence does not hold. Regarding the pair* $(\mathcal{C}; \mathcal{C}'_2)$*, we can go on and reach a solved form by applying* EQ-RIGHT-RIGHT *obtaining again two pairs of constraint systems. The first one will be of the form* $(\bot; \mathcal{C}'_3)$ *and the second one will contain two systems in which all the disequations are matched.*

## 5.2 How to apply the rules

We explain here how the transformation rules can be used on a pair of sets of constraint systems, assuming that all the constraint systems have the same structure. Actually, the basic idea is to apply the same transformation rule (with the same parameters) on each constraint system. Note that, the parameters of a transformation rule only depend on the structure of the underlying constraint system. Thanks to this, the simultaneous application of a transformation rule can be defined in a natural way. It consists of applying the same instance of the transformation rule on each constraint system that occurs in the two sets. So an application of a rule on a pair $(\Sigma, \Sigma')$ of sets of constraint systems will result in two pairs $(\Sigma_1, \Sigma'_1)$ and $(\Sigma_2, \Sigma'_2)$

Let $(\Sigma, \Sigma')$ be a pair of sets of constraint systems having the same structure. Let $\Sigma = \{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$, $\Sigma' = \{\mathcal{C}'_1, \ldots, \mathcal{C}'_\ell\}$, and R be an instance of a transformation rule. An application of R on the pair $(\Sigma, \Sigma')$ yields two pairs $(\Sigma_1, \Sigma'_1)$ and $(\Sigma_2, \Sigma'_2)$ such that:

- $\Sigma_i = \{\mathcal{C}_{i,1}, \ldots, \mathcal{C}_{i,k}\}$ for $i = 1, 2$; and
- $\Sigma'_i = \{\mathcal{C}'_{i,1}, \ldots, \mathcal{C}'_{i,\ell}\}$ for $i = 1, 2$.

where $(\mathcal{C}_{1,j}, \mathcal{C}_{2,j})$ (resp. $(\mathcal{C}'_{1,,j'}, \mathcal{C}'_{2,j'})$) is the pair of constraint systems obtained by applying R on $\mathcal{C}_j$ (resp. $\mathcal{C}'_{j'}$).

Actually, deciding symbolic equivalence between two sets raises two main issues:

- matching an existing solution from one set to the other;
- and deciding whether the two resulting frames are statically equivalence or not.

When checking static equivalence, we have to check that the same equalities hold in both resulting frames. So, in order to develop a simple test on leaves, it is important to gather the two resulting sets of constraint systems when the rule is used to check static equivalence on the resulting frames. This leads us to consider matrices of constraint systems and two kinds of applications for our rules: *internal* and *external*. An external application will apply a rule on the whole matrix while an internal application will apply a rule on one particular line (the same in both matrices) replacing this line with two new lines.

EXAMPLE 11. *Consider the following example where there is no deducibility constraint. The idea is to simply check whether static equivalence holds between sets of frames. Let* $\Phi_1 = \{ax_1, 1 \triangleright k, ax_2, 2 \triangleright a\}$ *and* $\Phi_2 = \{ax_1, 1 \triangleright k, ax_2 \triangleright \mathsf{senc}(a,k)\}$. *We consider the systems* $\mathcal{C}_i = (\Phi_i; \emptyset; \emptyset; \emptyset)$ *for* $i = 1, 2$. *Now, we want to check whether* $\{\mathcal{C}_1, \mathcal{C}_2\} \approx_s \{\mathcal{C}_2\}$. *Actually, symbolic equivalence does not hold since* $\Phi_1 \not\sim \Phi_2$. *However, applying* DEST, *the only rule that can be applied, will result in the pairs:*

$$(\ \{\bot, \mathcal{C}_2'\}\ ,\ \{\mathcal{C}_2'\}\ ) \qquad (\ \{\mathcal{C}_1'', \mathcal{C}_2''\}\ ,\ \{\ \mathcal{C}_2''\ \})$$

*where* $\mathcal{C}_1''$ *(resp.* $\mathcal{C}_2''$*) are obtained from* $\mathcal{C}_1$ *(resp.* $\mathcal{C}_2$*) by adding the non-deducibility constraints:*

- $\forall x, y \cdot [a \overset{?}{\neq} \mathsf{senc}(x,y) \vee \overset{?}{\forall} y]$ *in* $\mathcal{C}_1''$;
- $\forall x, y \cdot [\mathsf{senc}(a,k) \overset{?}{\neq} \mathsf{senc}(x,y) \vee \overset{?}{\forall} y]$ *in* $\mathcal{C}_2''$.

*Note that no transformation rule is applicable on this pair. It is easy to see that symbolic equivalence holds for the first pair. Actually, symbolic equivalence also holds on the second pair taking into account the non-deducibility constraints. However, we do not want to solve these non-deducibility constraints. Instead, we apply* DEST *internally, obtaining one leaf of the form:*

$$\begin{pmatrix} \bot & \mathcal{C}_2' \\ \mathcal{C}_1'' & \mathcal{C}_2'' \end{pmatrix} \ , \ \begin{pmatrix} \mathcal{C}_2' \\ \mathcal{C}_2'' \end{pmatrix}$$

*Then, on the leaves, we check that for each column of one matrix, there exist a column in the other matrix on which the constaint systems have the same status,* i.e. $\bot$ *or not. Here, this test will trivially fail on our unique leaf allowing us to conclude that the two original sets* $\{\mathcal{C}_1, \mathcal{C}_2\}$ *and* $\{\mathcal{C}_2\}$ *are* not *in symbolic equivalence.*

## 5.3 Strategy

Applying blindly the transformation rules does not always terminate (see [11] for an example). As in [11], to avoid non-terminating behaviors, we fix one of the constraint system, reduce it until reaching a "pre-solved form" (distinct variables in the right-hand sides of the deducibility constraints), and then move to the next system. Solving the second system does preserve the property on the first one. In this way we can reach constraint systems in "pre-solved" form.

This is not however sufficient: because of the disequations, it might be not so easy to decide the equivalence of such sets (or matrices) of constraint systems (see Example 8). Therefore, we further apply some transformation rules, that allow to simplify the disequalities. For this second phase, we only need the rules CONS, AXIOM, EQ-RIGHT-RIGHT. CONS is restricted to the situations in which there is a disequation $x \neq u$ and either $u$ is not a name and contains a name, or else $u$ is not a variable and contains a universally quantified variable. There is still one difficulty, because, as before, we may get a non terminating behavior.

EXAMPLE 12. *We consider a constraint system in "pre-solved" form such that:*

$$E = [x_1 \overset{?}{\neq} y \vee x_2 \overset{?}{\neq} a] \wedge y \overset{?}{\neq} \langle\langle x_1, x_2\rangle, b\rangle.$$

*For sake of simplicity, we do not described* $\Phi$ *and* $D$. *We simply assume that the frame contains the terms* $a$ *and* $b$. *First, we apply* AXIOM *on* $x_2$ *(with* $a$*), on one branch we will obtain* $x_1 \overset{?}{\neq} y \wedge y \overset{?}{\neq} \langle\langle x_1, a\rangle, b\rangle$. *Then, applying* CONS *twice, we obtain* $x_1 \overset{?}{\neq} \langle\langle y_1, y_2\rangle, y_3\rangle \wedge [y_1 \overset{?}{\neq} x_1 \vee y_2 \overset{?}{\neq} a \vee y_3 \overset{?}{\neq} b]$. *Lastly, applying* AXIOM *on* $y_3$ *(with* $b$*), we obtain*

$$x_1 \overset{?}{\neq} \langle\langle y_1, y_2\rangle, b\rangle \wedge [y_1 \overset{?}{\neq} x_1 \vee y_2 \overset{?}{\neq} a]$$

*getting back to the original set of disequations.*

We therefore use the following strategy:

$$((\text{CONS} + \text{EQ-RIGHT-RIGHT})!; \text{AXIOM}!)!$$

where the exclamation mark means "as long as possible". This allows us in particular to avoid the non-terminating behavior described in Example 12. Now, we claim that irreducible constraints contain only disequations $x \neq u$ where $u$ does not contain names or universally quantified variables and that the transformations are terminating.

The termination argument is as follows: EQ-RIGHT-RIGHT allows to "externalize" the disjunctions, splitting disjunctive constraints, each of which will appear in different matrices. Then, CONS will allow to decrease the heights of names and universally quantified variables. Finally, these measures may increase with AXIOM, but then the replacement substitutes a variable with support $i$ with terms that only contain variables, whose support is strictly smaller than $i$. Furthermore, all these rules will keep the deducibility constraints in pre-solved form.

## 5.4 Correctness

The transformation rules yield a finite tree labeled with pairs of matrices of constraint systems. As briefly explained in Example 11, our test on the leaves consists of checking that for each column in one matrix, there exists a column in the other matrix such that each constraint system has the same status. We say that $\mathsf{LeafTest}(M, M') = \mathsf{true}$ when this syntaxic test holds on the leaf $(M, M')$. Otherwise, we say that $\mathsf{LeafTest}(M, M') = \mathsf{false}$.

PROPOSITION 2. *Let* $(\Sigma_0, \Sigma_0')$ *be a pair of sets of constraint systems obtained by applying our transformation* $\mathbf{Tr}()$ *on sets of initial constraint systems, and consider a binary tree obtained by following the strategy described in Section 5.3.*

- soundness: *If all leaves of a tree are labeled with* $(M, M')$ *such that* $\mathsf{LeafTest}(M, M') = \mathsf{true}$, *then* $\Sigma_0 \approx_s \Sigma_0'$.
- completeness: *If* $\Sigma_0 \approx_s \Sigma_0'$, *then all leaves of a tree are labeled with* $(M, M')$ *with* $\mathsf{LeafTest}(M, M') = \mathsf{true}$.

The idea of the proof is to first analyse the structure of the leaves and then to show that our notion of equivalence is preserved through application of our transformation rules: for any transformation rule, if the two pairs of sets of constraint systems labeling the sons of a node are respectively in symbolic equivalence, then the same property holds for the father.

# 6. CONCLUSION

An Ocaml implementation of an early version of the procedure described in this paper has already been completed. This procedure extends [13] to set of constraints, including disequalities. Actually, checking symbolic equivalence between sets of constraint systems is quite efficient. However, the interleaving step, that is required for moving from symbolic equivalence to trace equivalence, is performed in a naive way and it appears that this step is expensive from the computation point of view. We tested this implementation on the private authentication protocol and the two versions of the e-passport protocol. Our implementation concludes within a few minutes for the private authentication protocol and the flawed version of the e-passport protocol (considering 2 sessions only). We also tried our implementation on the fixed version of the e-passport and it took more time (around 2 days).

In order to get an efficient procedure, it is necessary to come with some optimisations to reduce the search space and the number of interleavings. This problem is not specific to trace equivalence and has already been studied in the context of trace properties (*e.g.* [22]). However, discarding some "symbolic" interleavings appears to be challenging for equivalence-based properties. Finally, we would like to extend the method to other cryptographic primitives, typically blind signatures and zero-knowledge proofs.

# 7. REFERENCES


[1] M. Abadi and V. Cortier. Deciding knowledge in security protocols under equational theories. *Theoretical Computer Science*, 387(1-2):2–32, 2006.

[2] 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.

[3] M. Abadi and C. Fournet. Private authentication. *Theoretical Computer Science*, 322(3):427–476, 2004.

[4] M. Abadi and A. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proc. 4th Conference on Computer and Communications Security (CCS'97)*, pages 36–47. ACM Press, 1997.

[5] M. Arapinis, T. Chothia, E. Ritter, and M. Ryan. Analysing unlinkability and anonymity using the applied pi calculus. In *Proc. of 23rd IEEE Computer Security Foundations Symposium (CSF'10)*, pages 107–121. IEEE Computer Society Press, 2010.

[6] A. Armando et al. The AVISPA Tool for the automated validation of internet security protocols and applications. In *Proc. 17th Int. Conference on Computer Aided Verification (CAV'05)*, volume 3576 of *LNCS*, pages 281–285. Springer, 2005.

[7] M. Baudet. Deciding security of protocols against off-line guessing attacks. In *Proc. 12th Conference on Computer and Communications Security (CCS'05)*, pages 16–25. ACM Press, 2005.

[8] M. Baudet. *Sécurité des protocoles cryptographiques : aspects logiques et calculatoires*. Phd thesis, École Normale Supérieure de Cachan, France, 2007.

[9] B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *Proc. 14th Computer Security Foundations Workshop (CSFW'01)*, pages 82–96. IEEE Comp. Soc. Press, 2001.

[10] B. Blanchet, M. Abadi, and C. Fournet. Automated verification of selected equivalences for security protocols. *Journal of Logic and Algebraic Programming*, 75(1):3–51, 2008.

[11] V. Cheval, H. Comon-Lundh, and S. Delaune. Automating security analysis: symbolic equivalence of constraint systems. In *Proc. 5th International Joint Conference on Automated Reasoning (IJCAR'10)*, volume 6173 of *LNAI*, pages 412–426. Springer-Verlag, 2010.

[12] Y. Chevalier and M. Rusinowitch. Decidability of symbolic equivalence of derivations. *Journal of Automated Reasoning*, 2011. To appear.

[13] H. Comon-Lundh and S. Delaune. The finite variant property: How to get rid of some algebraic properties. In *Proc. 16th International Conference on Rewriting Techniques and Applications (RTA'05)*, LNCS, pages 294–307. Springer, 2005.

[14] V. Cortier and S. Delaune. A method for proving observational equivalence. In *Proc. 22nd Computer Security Foundations Symposium (CSF'09)*, pages 266–276. IEEE Comp. Soc. Press, 2009.

[15] S. Delaune, S. Kremer, and M. D. Ryan. Symbolic bisimulation for the applied pi-calculus. In *Proc. 27th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'07)*, pages 133–145, 2007.

[16] S. Delaune, S. Kremer, and M. D. Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17(4):435–487, July 2009.

[17] D. Dolev and A. C. Yao. On the security of public key protocols. In *Proc. 22nd Symposium on Foundations of Computer Science (FCS'81)*, pages 350–357. IEEE Computer Society Press, 1981.

[18] L. Durante, R. Sisto, and A. Valenzano. Automatic testing equivalence verification of spi calculus specifications. *ACM Transactions on Software Engineering and Methodology*, 12(2):222–284, 2003.

[19] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Workshop on Formal Methods and Security Protocols*, 1999.

[20] H. Hüttel. Deciding framed bisimulation. In *Proc. 4th Int. Workshop on Verification of Infinite State Systems (INFINITY'02)*, pages 1–20, 2002.

[21] J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proc. 8th ACM Conference on Computer and Communications Security (CCS'01)*. ACM Press, 2001.

[22] S. Mödersheim, L. Viganò, and D. A. Basin. Constraint differentiation: Search-space reduction for the constraint-based analysis of security protocols. *Journal of Computer Security*, 18(4):575–618, 2010.

[23] R. Ramanujam and S. Suresh. Tagging makes secrecy decidable for unbounded nonces as well. In *Proc. 23rd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'03)*, 2003.

[24] M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *Proc. 14th Computer Security Foundations Workshop (CSFW'01)*, pages 174–190. IEEE Comp. Soc. Press, 2001.

[25] A. Tiu and J. E. Dawson. Automating open bisimulation checking for the spi calculus. In *Proc. 23rd IEEE Computer Security Foundations Symposium (CSF'10)*, pages 307–321. IEEE Computer Society Press, 2010.