Automated verification of equivalence properties of cryptographic protocols

Rohit Chadha, Stefan Ciobaca, Steve Kremer

Automated verification of equivalence properties of cryptographic protocols*

Rohit Chadha[1], Ştefan Ciobâcă[1], and Steve Kremer[1,2]

[1] LSV, ENS Cachan & CNRS & INRIA
[2] INRIA Nancy - Grand-Est

Abstract. Indistinguishability properties are essential in formal verification of cryptographic protocols. They are needed to model anonymity properties, strong versions of confidentiality and resistance to offline guessing attacks, and can be conveniently modeled using process equivalences. We present a novel procedure to verify equivalence properties for bounded number of sessions. Our procedure is able to verify trace equivalence for determinate cryptographic protocols. On determinate protocols, trace equivalence coincides with observational equivalence which can therefore be automatically verified for such processes. When protocols are not determinate our procedure can be used for both under- and over-approximations of trace equivalence, which proved successful on examples. The procedure can handle a large set of cryptographic primitives, namely those which can be modeled by an optimally reducing convergent rewrite system. Although, we were unable to prove its termination, it has been implemented in a prototype tool and has been effectively tested on examples, some of which were outside the scope of existing tools.

1 Introduction

Cryptographic protocols are distributed programs which rely on the use of cryptography to secure electronic transactions such as those that arise in electronic commerce and wireless communication. They are also being applied in new domains such as in Internet voting—legally binding political elections in Estonia, Norway and Switzerland offer the possibility for Internet voting in 2011. This has led to increasing demands on the complexity of desired security properties, leading to more complex cryptographic protocols. Given the socio-economic-political consequences and the history of incorrect design of cryptographic protocols, the need for formal proofs of correctness of protocols has been widely recognized. Formal reasoning about cryptographic protocols is challenging as one has to reason against all potentially malicious behavior—all communication between protocol participants is assumed to be under the control of an adversary.

In order to make the task of formal analysis amenable to automation, usually the assumption of black-box cryptography and unbounded computational power

* This work was partially supported by ANR projects ARA SESUR AVOTÉ and JCJC VIP no 11 JS02 006 01 and the ERC grant agreement n^o 258865, project ProSecure.

of the adversary is made. This adversarial model is often called the Dolev-Yao model and is derived from Dolev and Yao's seminal paper [29]. It has proved extremely successful, and there are several automated tools [10, 6, 31] that can automatically check trace-properties such as (weak forms of) confidentiality and authentication. While trace-based properties are certainly important, many crucial security properties can only be expressed in terms of *indistinguishability* (or equivalence). They include strong flavors of confidentiality [11]; resistance to guessing attacks in password based protocols [8]; and anonymity properties in private authentication [3], electronic voting [26, 7], vehicular networks [24] and RFID protools [5, 15]. More generally, indistinguishability allows to model security by the means of ideal systems, which are correct by construction [4, 25]. Indistinguishability properties of cryptographic protocols are naturally modeled by the means of *observational* and *testing equivalences* in cryptographic extensions of process calculi, e.g., the spi [4] and the applied-pi calculus [2]. While we have good tools for automated verification of trace properties, the situation is different for indistinguishability properties.

State-of-the-art. Hüttel [34] showed undecidability of observational equivalence in the spi calculus, even for the finite control fragment, as well as decidability for the finite, i.e., replication-free, fragment of the spi calculus. The decidability result however only holds for a fixed set of cryptographic primitives and does not yield a practical algorithm. Current results [12] allow to approximate observational equivalence for an unbounded number of sessions. However, this approximation does not suffice to conclude for many applications, *e.g.*, [26, 5]. Our approach overcomes these limitations for some applications in [26]. We still cannot conclude for the e-passport example in [5], albeit for a different reason: our procedure does not currently handle else branches in protocols.

Symbolic bisimulations have also been devised for the spi [14, 13, 39] and applied pi calculus [27, 35] to avoid unbounded branching due to adversary inputs. However, only [27, 39] and [14] yield a decision procedure, again only approximating observational equivalence. The results of [27] have been further refined to show a decision procedure on a restricted class of *simple* processes [23]. They rely on a procedure deciding the equivalence of constraint systems, introduced by Baudet [8], for the special case of verifying the existence of guessing attacks. Baudet's procedure allows arbitrary cryptographic primitives that can be modeled as a subterm convergent rewrite systems [1]. An alternate procedure achieving the same goal was proposed by Chevalier and Rusinowitch [19]. However, both procedures are highly non-deterministic and do not yield a reasonable algorithm that could be implemented. Therefore, Cheval *et al.* [17] have designed a new procedure and a prototype tool to decide the equivalence of constraint systems, but only for a fixed set of primitives. Tools have also been implemented for checking testing equivalence [30], open bisimulation [39] and trace equivalence [18] for a bounded number of sessions but again only for a limited set of primitives. One may note that [18] is the only decision procedure to consider negative tests (else branches), crucial in several case studies [5, 3].

Our contribution. We introduce a new procedure for verifying equivalence properties for processes specified in a cryptographic process calculus (without replication). Our main contributions are as follows.

- Our procedure checks for two equivalences which over- and under-approximate the standard notion of trace equivalence $\approx_t$ for cryptographic protocols: the under-approximation can be used to prove protocols correct while the over-approximation can be used to rule out incorrect protocols.
- Cortier and Delaune [23] have shown that observational equivalence coincides with $\approx_t$ for the class of *determinate* processes. They also give a decision procedure for a strict sub-class of determinate processes, namely, *simple* processes. We show that for determinate processes the coarser relation coincides with $\approx_t$, and our procedure can be used to verify observational equivalence for the whole class of determinate processes.
- A novelty of our procedure is that it is based on a *fully abstract* modeling of symbolic traces in *first-order Horn clauses*. This is in contrast to the constraint-solving techniques employed in [39, 17, 18, 8, 19] for verifying under-approximations of observational equivalence. Techniques based on Horn clauses have been extensively used, e.g., in [10, 40, 33], for an unbounded number of sessions. Of these tools, only ProVerif [10, 12] can verify an equivalence property, which is an under-approximation of observational equivalence. Horn clause modeling of an unbounded number of sessions of security protocols may allow false attacks. In contrast, we show our modeling of a bounded number of sessions for determinate protocols to be precise.
- Our modeling is fully abstract for arbitrary cryptographic primitives that can be modeled as a convergent rewrite system which has the *finite variant property*. Not only this strictly includes the class of primitives that can be modeled as subterm convergent rewrite systems, but this also allows us to handle a larger class of cryptographic primitives than [39, 17, 18, 8, 19, 10]. For example, this allows us to handle trapdoor commitment as used by Okamoto for electronic voting in [38]. Although we were unable to prove termination of our procedure, we conjecture it to terminate for the class of cryptographic primitives that can be modeled as subterm convergent rewrite systems. Our conjecture is supported by experimental evidence.
- Our procedure is implemented in the AKiSs (Active Knowledge in Security protocols) prototype tool and used among others to give the first automated proof of anonymity for the electronic voting protocol presented in [32].

Technical proofs are given in an accompanying technical report [16].

2 Preliminaries

Terms. Let $\mathcal{F}$ be a signature, i.e., a finite set of function symbols and ar a function that assigns to each function symbol a natural number, its arity. A function symbol of arity 0 is called a *constant*. Given a set of *atoms* $\mathcal{A}$ and a signature $\mathcal{F}$, we denote by $\mathcal{T}_{\mathcal{F},\mathcal{A}}$ the set of terms built inductively from $\mathcal{A}$ by

applying functions symbols in $\mathcal{F}$. Given sets of atoms $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n$, we denote the set $\mathcal{T}_{\mathcal{F},\cup_{1\leq i\leq n}\mathcal{A}_i}$ by $\mathcal{T}_{\mathcal{F},\mathcal{A}_1,\mathcal{A}_2,\ldots\mathcal{A}_n}$. We assume that we have the following countably infinite pairwise disjoint sets: a set $\mathcal{N}$ of *private names*, $\mathcal{M}$ of *public names*, a set $\mathcal{C}$ of *public channel names*, a set $\mathcal{W}$ of *parameters*, and a set $\mathcal{X}$ of *message variables*. Intuitively, elements of the set $\mathcal{N}$ represent nonces generated by honest principals of a protocol, elements of $\mathcal{M}$ represent nonces available both to the adversary and to the honest participants and elements of $\mathcal{C}$ represent names of public channels (e.g. the name of a public network). Elements of $\mathcal{W}$ are pointers used by the adversary to refer to messages output by the honest participants in a protocol. We fix an enumeration $w_1, w_2, \ldots$ of the elements of $\mathcal{W}$. We let x, y, z range over $\mathcal{X}$. We also define the following set of terms:

- Terms denotes the set of all terms $\mathcal{T}_{\mathcal{F},\mathcal{N},\mathcal{M},\mathcal{W},\mathcal{X}}$.
- $\mathsf{Messages}$ denotes the set of *messages* $\mathcal{T}_{\mathcal{F},\mathcal{N},\mathcal{M}}$.
- $\mathsf{SMessages}$ denotes the set of *symbolic messages* $\mathcal{T}_{\mathcal{F},\mathcal{N},\mathcal{M},\mathcal{X}}$.

If t is a term, we denote by $vars(t)$ the set of variables appearing in t, by $names(t)$ the set of names (public or private) appearing in t. The functions $vars$, $names$ are extended to sequences and sets of terms as expected.

Example 1. Consider the signature $\mathcal{F} = \{\mathsf{enc}, \mathsf{dec}, \mathsf{pair}, \mathsf{fst}, \mathsf{snd}\}$. The term $t = \mathsf{pair}(\mathsf{enc}(a, k_1, r_1), \mathsf{enc}(b, k_2, r_2))$ models the pair of the asymmetric encryptions of public names a and b with keys k_1, resp. k_2 and randomness r_1, resp. r_2.

A substitution is a partial function $\sigma : \mathcal{W} \cup \mathcal{X} \to \mathsf{Terms}$. We restrict substitutions to map elements of $\mathcal{W}$ to elements of $\mathsf{Messages}$ and elements of $\mathcal{X}$ to elements of $\mathsf{SMessages}$. The domain of σ shall be denoted by $dom(\sigma)$. We denote by $\sigma[X]$ the substitution whose domain is restricted to X. We only consider substitutions with finite domains. As usual, a substitution extends homomorphically to terms and we write $t\sigma$ for the term obtained by applying σ to t.

Rewriting and unification. Two terms s and t are (syntactically) *unifiable* if there exists a substitution σ such that $s\sigma = t\sigma$. We denote by $\mathrm{mgu}(s, t)$ their most general unifier. We assume that the reader is familiar with basic notions of rewriting and only briefly introduce our notations. A rewrite system R is a set of rewrite rules of the form $\ell \to r$ where $\ell, r \in \mathsf{Terms}$, $names(l, r) = \emptyset$ and $vars(r) \subseteq (\ell)$. We write $t \to_{\mathsf{R}} u$ when a term t can be rewritten in one step to u. $\to_{\mathsf{R}}^*$ denotes the transitive and reflexive closure of $\to_{\mathsf{R}}$. We only consider convergent rewrite systems and denote by $t{\downarrow_{\mathsf{R}}}$ the normal form of a term t. Two terms s and t are said to be equal modulo R, written $s =_{\mathsf{R}} t$, if $s{\downarrow_{\mathsf{R}}} = t{\downarrow_{\mathsf{R}}}$. Given a substitution σ, $\sigma{\downarrow_R}$ is the substitution such that $dom(\sigma{\downarrow_{\mathsf{R}}}) = dom(\sigma)$ and for all $u \in dom(\sigma)$, $\sigma{\downarrow_R}(u) = \sigma(u){\downarrow_{\mathsf{R}}}$. We shall omit R when clear from the context.

Example 2. Let $\mathcal{F}$ be the signature in Example 1. Consider the rewrite system $\mathsf{R} = \{\mathsf{dec}(\mathsf{enc}(x, y, z), y) \to x, \mathsf{fst}(\mathsf{pair}(x, y)) \to x, \mathsf{snd}(\mathsf{pair}(x, y)) \to y\}$. The first rule models that a message can be decrypted, provided decryption uses the same key (represented by variable y) as encryption. The last two rules model projection of the first and second component of a pair. We have that $t = \mathsf{fst}(\mathsf{pair}(\mathsf{dec}(\mathsf{enc}(a, k, r), k), b)) \to_{\mathsf{R}} \mathsf{fst}(\mathsf{pair}(a, b)) \to_{\mathsf{R}} a = t{\downarrow_{\mathsf{R}}}$.

We recall the notion of complete set of variants for a convergent rewrite system [22]:

Definition 1. *A set of substitutions* variants$(t_1, \ldots, t_k)$ *is called a* complete set of variants *of terms* $t_1, \ldots, t_k$ *if for any substitution* ω *there exist* $\sigma \in$ variants$(t_1, \ldots, t_k)$ *and a substitution* τ *such that for all* $1 \leq j \leq k$ *we have that* $\omega[vars(t_j)]\downarrow = (\sigma\downarrow\tau)[vars(t_j)]$ *and* $(t_j\omega)\downarrow = (t_j\sigma)\downarrow\tau$.

Intuitively, the set of variants of t represents a pre-computation such that any instance of t in normal form is *syntactically* equal to an instance of $t\sigma_i\downarrow$ for some i, without the need to apply further rewrite steps. A rewrite system has the *finite variant property* if for any finite sequence of terms a finite, complete set of variants exists. An algorithm for computing complete sets of variants which is correct whenever the rewrite system is *optimally reducing* [37] is presented in [21]. Optimally reducing rewrite systems include subterm convergent systems [1] (and hence the classical Dolev Yao theories for encryption, signatures and hash functions), as well as a theory for modeling blind signatures. Complete sets of variants can be used to compute finite complete sets of unifiers modulo R [21], which are formally defined in [16] and denoted by $\mathrm{mgu}_{\mathsf{R}}$. We assume, henceforth, that rewrite systems in this paper have the finite variant property.

Frames, deducibility and static equivalence. We will use the notion of a *frame* [2] to represent messages which have been recorded by an attacker.

Definition 2. *A* frame φ *is a substitution* $\{w_1 \mapsto t_1, \ldots, w_n \mapsto t_n\}$ *where* $t_i \in$ Messages *(*$1 \leq i \leq n$*).*

Please note, in our definition, every frame φ with $|dom(\varphi)| = n$ has $dom(\varphi) = \{w_1, \ldots, w_n\}$. The set of all frames is denoted as Frames. The adversary can use the messages learnt from the run of a protocol to construct new messages. This is modeled as the deducibility relation.

Definition 3. *Any term in* $\mathcal{T}_{\mathcal{F},\mathcal{M},\mathcal{W}}$ *is said to be a recipe. We say that* a message t is deducible from φ with a recipe r *(written as* $\varphi \vdash^r t$*) if* $t \in$ Messages *and* $r\varphi =_{\mathsf{R}} t$. *We write* Recipes *for the set* $\mathcal{T}_{\mathcal{F},\mathcal{M},\mathcal{W}}$.

Example 3. Consider the signature $\mathcal{F}$ and the rewrite system R in Example 2. Let $\varphi = \{w_1 \mapsto \mathsf{enc}(s, k, r), w_2 \mapsto k\}$ where $s, k, r \in \mathcal{N}$ are private names. We have that $\varphi \vdash^{\mathsf{dec}(w_1,w_2)} s$. Note that $\mathsf{dec}(w_1, k) \notin$ Recipes as $k \in \mathcal{N}$. If s were public instead of being private (ie, $s \in \mathcal{M}$ instead of $s \in \mathcal{N}$) then we also have that $\varphi \vdash^s s$; as public names are always deducible.

Static equivalence captures indistinguishability of sequences of messages:

Definition 4. *Let* $r_1, r_2 \in$ Recipes. *A* test $r_1 \stackrel{?}{=} r_2$ holds *in a frame* φ *(written* $(r_1 = r_2)\varphi$*) if* $\varphi \vdash^{r_1} t$ *and* $\varphi \vdash^{r_2} t$ *for some* t, i.e., r_1 *and* r_2 *are recipes for the same term in* φ.

A frame φ_1 *is* statically included *in* φ_2 *(written* $\varphi_1 \sqsubseteq_s \varphi_2$*) iff for all* $r_1, r_2 \in$ Recipes *we have that* $(r_1 = r_2)\varphi_1$ *implies* $(r_1 = r_2)\varphi_2$. *Two frames* φ_1 *and* φ_2 *are* statically equivalent *(written* $\varphi_1 \approx_s \varphi_2$*) iff* $\varphi_1 \sqsubseteq_s \varphi_2$ *and* $\varphi_2 \sqsubseteq_s \varphi_1$.

Example 4. Let $a, b \in \mathcal{M}$ and $r, k, k' \in \mathcal{N}$. We have that $\{w_1 \mapsto \mathsf{enc}(a, k, r), w_2 \mapsto k\} \not\approx_s \{w_1 \mapsto \mathsf{enc}(b, k, r), w_2 \mapsto k\}$ because the test $(dec(w_1, w_2) = a)$ distinguishes the two frames. However, $\{w_1 \mapsto \mathsf{enc}(a, k, r), w_2 \mapsto k'\} \approx_s \{w_1 \mapsto \mathsf{enc}(b, k, r), w_2 \mapsto k'\}$. Moreover, we have that $\{w_1 \mapsto a, w_2 \mapsto b\} \sqsubseteq_s \{w_1 \mapsto a, w_2 \mapsto a\}$ while $\{w_1 \mapsto a, w_2 \mapsto a\} \not\sqsubseteq_s \{w_1 \mapsto a, w_2 \mapsto b\}$.

3 A cryptographic process calculus

We model cryptographic protocols using a simple process calculus which has similarities with the applied pi-calculus [2].

Syntax. We model a bounded number of instances of a cryptographic protocol as a *finite* set of traces. Traces are defined using sequences of *actions* generated by the following grammar:

$$a ::= \mathbf{in}(c, x) \;\mid\; \mathbf{out}(c, t) \;\mid\; [s \stackrel{?}{=} t]$$

where $x \in \mathcal{X}, s, t \in \mathsf{SMessages}, c \in \mathcal{C}$. A *trace* T is a sequence of actions $T = a_1.a_2.\ldots.a_n$. As usual, a receive action $\mathbf{in}(c, x)$ acts as a binding construct for x. We assume the usual definitions of free and bound variables for traces. We also assume that each variable is bound at most once. A trace is *ground* if it does not contain any free variables. The set of ground traces shall be represented as $\mathsf{GndTraces}$. A set of traces $P = \{T_1, \ldots, T_n\}$ is said to be a *process*. A process is ground if all of its traces are ground. We identify traces with singleton processes.

Remark 1. We do not have an ν operator: the binding happens implicitly by the use of private names in $\mathcal{N}$. We have also not explicitly included the parallel operator | and the choice operator +. One could include these and generate the corresponding set of traces. Thus, there is no loss in expressivity. However, an explicit enumeration of the traces can result in an exponential number of traces.

Semantics. The semantics of a process is defined using the semantics of its traces. The semantics of a trace is given in terms of a labeled transition system T. We assume that all interactions between protocol participants are mediated by the adversary. The labeled transition system records the interaction of the protocol participants with the adversary. The set of labels of T is defined using the set $\mathsf{Recipes}$. Recall that the set $\mathsf{Recipes}$ is the set $\mathcal{T}_{\mathcal{F},\mathcal{M},\mathcal{W}}$ (see Section 2). The set of labels, Labels, is $\{\, \mathbf{in}(c, r), \mathbf{out}(c), \mathbf{test} \mid r \in \mathsf{Recipes}, c \in \mathcal{C} \,\}$.

The labeled transition system T is a subset of $(\mathsf{GndTraces} \times \mathsf{Frames}) \times \mathsf{Labels} \times (\mathsf{GndTraces} \times \mathsf{Frames})$. We write $(T, \varphi) \xrightarrow{\ell} (T', \varphi')$ whenever $((T, \varphi), \ell, (T', \varphi')) \in \mathsf{T}$. The frame in the transition system is used to record the messages that the protocol participants have sent in the past. The relation $\xrightarrow{\ell}$ is defined as follows:

$$\textsc{Receive}\ \frac{\varphi \vdash^r t}{(\mathbf{in}(c, x).T, \varphi) \xrightarrow{\mathbf{in}(c,r)} (T\{x \mapsto t\}, \varphi)} \qquad \textsc{Test}\ \frac{s =_{\mathsf{R}} t}{([s \stackrel{?}{=} t].T, \varphi) \xrightarrow{\mathbf{test}} (T, \varphi)}$$

$$\textsc{Send}\ \frac{}{(\mathbf{out}(c, t).T, \varphi) \xrightarrow{\mathbf{out}(c)} (T, \varphi \cup \{w_{|dom(\varphi)|+1} \mapsto t\})}$$

The label $\mathbf{in}(c, r)$ indicates a message sent by the adversary over the channel c and r is the recipe that adversary uses to create this message. The label $\mathbf{out}(c)$ indicates a message sent over the public channel c and transition rule SEND records the message sent in the frame. Finally, the rule Test is an internal action.

We write $(T, \varphi) \stackrel{\ell}{\Rightarrow} (T', \varphi')$ when either $(T, \varphi) \xrightarrow{\mathbf{test}^*, \ell, \mathbf{test}^*} (T', \varphi')$ and $\ell \neq \mathbf{test}$ or $(T, \varphi) \xrightarrow{\mathbf{test}^*} (T', \varphi')$ and $\ell = \mathbf{test}$, where $\mathbf{test}^*$ denotes an arbitrary number of **test** actions. We also write $(T_0, \varphi_0) \xrightarrow{\ell_1, \ldots, \ell_n} (T_n, \varphi_n)$ when $(T_0, \varphi_0) \xrightarrow{\ell_1} (T_1, \varphi_1) \ldots \xrightarrow{\ell_n} (T_n, \varphi_n)$ (and similarly for the $\Rightarrow$ relation) and say that $\ell_1 \ldots \ell_n$ is a *run* of (T_0, φ_0). If P is a process, we write $(P, \varphi) \xrightarrow{\ell_1, \ldots, \ell_n} (T', \varphi')$ (resp. $\stackrel{\ell_1, \ldots, \ell_n}{\Longrightarrow} (T', \varphi')$) if there exists a trace $T \in P$ such that $(T, \varphi) \xrightarrow{\ell_1, \ldots, \ell_n} (T', \varphi')$ (resp. $(T, \varphi) \stackrel{\ell_1, \ldots, \ell_n}{\Longrightarrow} (T', \varphi')$).

Process equivalences. We will now define different flavors of trace equivalence which will be useful in this paper. We first recall the standard definition of trace equivalence in cryptographic process algebras.

Definition 5. ***(Trace equivalence)*** *A ground process P is said to be* trace-included *in a ground process Q (written $P \sqsubseteq_t Q$) if whenever $(P, \emptyset) \stackrel{\ell_1, \ldots, \ell_n}{\Longrightarrow} (T, \varphi)$ then there exist T', φ' such that $(Q, \emptyset) \stackrel{\ell_1, \ldots, \ell_n}{\Longrightarrow} (T', \varphi')$ and $\varphi \approx_s \varphi'$. Two processes P and Q are* trace-equivalent *(written $P \approx_t Q$) if $P \sqsubseteq_t Q$ and $Q \sqsubseteq_t P$.*

We will also define two other notions of trace equivalence, one coarser and one more fine-grained.

Definition 6. *Given ground processes P and Q, we say that $P \sqsubseteq_{ct} Q$ if whenever $(P, \emptyset) \stackrel{\ell_1, \ldots, \ell_n}{\Longrightarrow} (T, \varphi)$ then there exist T', φ' such that $(Q, \emptyset) \stackrel{\ell_1, \ldots, \ell_n}{\Longrightarrow} (T', \varphi')$ and $\phi \sqsubseteq_s \phi'$. We say that $P \approx_{ct} Q$ if $P \sqsubseteq_{ct} Q$ and $Q \sqsubseteq_{ct} P$.*

The following example illustrates the difference between $\approx_t$ and $\approx_{ct}$.

Example 5. Let P and Q be the ground processes defined as follows: $P = \{\, \mathbf{out}(c, a).\mathbf{out}(c, a) \,\}$ and $Q = \{\, \mathbf{out}(c, a).\mathbf{out}(c, a), \mathbf{out}(c, a).\mathbf{out}(c, b) \,\}$. Clearly $P \sqsubseteq_{ct} Q$. Observe also that $Q \sqsubseteq_{ct} P$. This is because $\{w_1 \mapsto a, w_2 \mapsto b\} \sqsubseteq_s \{w_1 \mapsto a, w_2 \mapsto a\}$. Thus, $P \approx_{ct} Q$. But $P \not\approx_t Q$.

We show, however, that these two notions coincide for the class of *determinate processes*. In the context of the applied pi calculus determinate processes were previously studied by Cortier and Delaune in [23].

Definition 7. ***(Determinate process)*** *A ground process P is determinate if whenever $(P, \emptyset) \stackrel{\ell_1, \ldots, \ell_n}{\Longrightarrow} (T, \varphi)$ and $(P, \emptyset) \stackrel{\ell_1, \ldots, \ell_n}{\Longrightarrow} (T', \varphi')$ then $\varphi \approx_s \varphi'$.*

Intuitively, determinate processes are processes in which the adversary's static knowledge at any instance is completely determined by its past interaction with the protocol participants. Note that any ground trace is determinate.

As already mentioned above, it was demonstrated in [23] that trace equivalence coincides with observational equivalence for determinate processes. We show that $\approx_t$ and $\approx_{ct}$ also coincide for this class of processes.

Theorem 1. *If P and Q are ground processes then $P \approx_t Q$ implies $P \approx_{ct} Q$. Furthermore, if P and Q are determinate, then $P \approx_{ct} Q$ implies $P \approx_t Q$.*

We introduce a more fine-grained notion of trace equivalence, denoted $\approx_{ft}$.

Definition 8. *Given ground processes P and Q, we say that $P \sqsubseteq_{ft} Q$ whenever for all trace $T \in P$ there exists a trace $T' \in Q$ such that $T \approx_t T'$. We say that $P \approx_{ft} Q$ if $P \sqsubseteq_{ft} Q$ and $Q \sqsubseteq_{ft} P$.*

It follows directly form the definition that $\approx_{ft} \subset \approx_t$. The difference between these two relations is illustrated by the following example.

Example 6. Let P and Q be ground processes defined as follows:

$$\begin{aligned} P = \{\ & \mathbf{out}(c, enc(a,k)).\mathbf{out}(c, enc(b,k)).\mathbf{in}(c,x).[x = enc(a,k)].\mathbf{out}(c,k), \\ & \mathbf{out}(c, enc(a,k)).\mathbf{out}(c, enc(b,k)).\mathbf{in}(c,x).[x = enc(b,k)].\mathbf{out}(c,k)\} \\ Q = \{\ & \mathbf{out}(c, enc(a,k)).\mathbf{out}(c, enc(b,k)).\mathbf{in}(c,x).[x = enc(dec(x,k),k)].\mathbf{out}(c,k)\} \end{aligned}$$

where $k \in \mathcal{N}$ is a private name and a, b are constants. The test $x = enc(dec(x,k),k)$ simply checks whether x is an encryption with key k. It is not difficult to see that $P \approx_t Q$ but $P \not\approx_{ft} Q$.

Our procedure is able to check $\approx_{ct}$ (and hence $\approx_t$) for determinate processes. For non-determinate processes, we can check $\approx_{ft}$ and an over-approximation of $\approx_{ct}$ (see [16] for details) in order to under- and over-approximate $\approx_t$: as traces are determinate a procedure for checking $\approx_{ct}$ can be used to verify $\approx_{ft}$.

4 Modeling traces as Horn clauses

Our procedure is based on a fully abstract modeling of a trace into first-order Horn clauses. We give the details of this modeling; we start by giving some definitions that we need for defining the predicates used in the logic.

Symbolic labels and symbolic runs. We define the set of *symbolic labels* as

$$\mathsf{SLabels} = \{\mathbf{in}(c,t), \mathbf{out}(c), \mathbf{test} \mid t \in \mathsf{SMessages}, c \in \mathcal{C}\}$$

and the set of *symbolic runs* as the set of finite sequences of symbolic labels (see Figure 1). The empty sequence is denoted by ϵ. We will often be lazy and write (empty space) for ϵ. Intuitively, a symbolic label stands for a set of possible labels, and a symbolic run stands for a set of possible runs of the protocol.

Symbolic Recipes. We assume a set $\mathcal{Y}$ of *recipe variables* disjoint from $\mathcal{X}$. The set of terms $\mathcal{T}_{\mathcal{F},\mathcal{M},\mathcal{W},\mathcal{Y}}$ shall be called *symbolic recipes* and denoted by $\mathsf{SRecipes}$. We use capital letters X, Y, Z to range over $\mathcal{Y}$. Intuitively, a symbolic recipe stands for a set of recipes. We can extend the definition of substitutions to include variables from $\mathcal{Y}$ in its domain: we only consider substitutions that map variables in $\mathcal{Y}$ to $\mathsf{SRecipes}$. A ground substitution must map variables in $\mathcal{Y}$ to $\mathsf{Recipes}$. The notions of mgu and $\mathsf{mgu}_{\mathsf{R}}$ is extended to symbolic recipes as expected.

Predicates. The predicates used in our modeling and the semantics of the predicates are given in Figure 1. The predicates are interpreted over a triple– a trace T, a frame φ and a substitution σ. We have four kinds of predicates, all of which have a symbolic run as an argument. Intuitively, the *reachability predicate* r_w says that each run represented by w is possible. The intruder knowledge predicate $\mathsf{k}_w(R,t)$ says that whenever a run represented by w happens, the (symbolic) message t can be constructed by the intruder using the (symbolic) recipe R. The identity predicate $\mathsf{i}_w(R,R')$ says that whenever the (symbolic) run SR happens, the (symbolic) recipes R and R' are recipes for the same (symbolic) term. The reachable identity predicate $\mathsf{ri}_w(R,R')$ is a short form for the conjunction of the predicates r_w and $\mathsf{i}_w(R,R')$.

Formulas and statements. We consider first-order formulas built using the above predicates and the usual connectives (conjunction, disjunction, negation, implication, existential and universal quantification). As in the case of predicates, a formula is interpreted over a triple consisting of a trace T, a frame φ and a substitution σ; and the semantics is defined as expected. For ground formulas we do not need the substitution σ and when a formula f is ground we simply write $(T,\varphi) \models f$ to denote that this formula holds for (T,φ). If moreover, $dom(\varphi) = \emptyset$, we simply write $T \models f$ for $(T,\emptyset) \models f$.

We now identify a subset of the formulas, which we shall call statements. Statements shall take the form of Horn clauses.

Definition 9. *A* statement *is a Horn clause of the form* $H \Leftarrow B_1, \ldots, B_n$ *where:*

1. $H \in \{\mathsf{r}_{\ell_1,\ldots,\ell_k}, \mathsf{k}_{\ell_1,\ldots,\ell_k}(R,t), \mathsf{i}_{\ell_1,\ldots,\ell_k}(R,R'), \mathsf{ri}_{\ell_1,\ldots,\ell_k}(R,R')\}$
2. *For each* $1 \leq i \leq n, B_i = \mathsf{k}_{\ell_1,\ldots,\ell_{j_i}}(X_i,t_i)$

for some $\ell_1,\ldots,\ell_k \in \mathsf{SLabels}$, $t \in \mathsf{SMessages}$, $R, R' \in \mathsf{SRecipes}$, $j_i \leq k$, $t_1,\ldots,t_n \in \mathsf{SMessages}$ *and* $X_1,\ldots,X_n \in \mathcal{Y}$. *Furthermore* $X_1,\ldots,X_n$ *are distinct variables and if* $H = \mathsf{k}_{\ell_1,\ldots,\ell_k}(R,t)$ *then* $vars(t) \subseteq vars(t_1,\ldots,t_n)$.

As usual, we implicitly assume that in a Horn clause all variables are universally quantified. Hence, all statements are closed formulas.

The set of seed statements. Our procedure is based on a fully abstract modeling of a trace in first-order Horn clauses. In this section, given a trace T we define a set of statements $\mathsf{seed}(T)$ that serve as a starting point for the modeling. We also establish that $\mathsf{seed}(T)$ is a sound and (partially) complete abstraction of the trace T. In order to formally define $\mathsf{seed}(T)$, we start by fixing some conventions.

Symbolic Runs ($\ell \in \mathsf{SLabels}$):
$u, v, w := \epsilon \mid \ell, w$

Predicates ($w \in \mathsf{SRuns}, R \in \mathsf{SRecipes}, t \in \mathsf{SMessages}$):

r_w	(Reachability predicate)
$\mathsf{k}_w(R,t)$	(Intruder knowledge predicate)
$\mathsf{i}_w(R,R')$	(Identity predicate)
$\mathsf{ri}_w(R,R')$	(Reachable identity predicate)

Semantics ($\ell_i \in \mathsf{SLabels}, R \in \mathsf{SRecipes}, t \in \mathsf{SMessages}, T \in \mathsf{GndTraces}, \varphi \in \mathsf{Frames}$, σ a ground substitution):

$(T,\varphi_0,\sigma) \models \mathsf{r}_{\ell_1,\ldots,\ell_i}$	if $(T,\varphi_0) \xrightarrow{L_1} (T_1,\varphi_1) \xrightarrow{L_2} \ldots \xrightarrow{L_n} (T_n,\varphi_n)$ such that $\ell_i\sigma =_{\mathsf{R}} L_i\varphi_{i-1}$ for all $1 \le i \le n$
$(T,\varphi_0,\sigma) \models \mathsf{k}_{\ell_1,\ldots,\ell_i}(R,t)$	if when $(T,\varphi_0) \xrightarrow{L_1} (T_1,\varphi_1) \xrightarrow{L_2} \ldots \xrightarrow{L_n} (T_n,\varphi_n)$ such that $\ell_i\sigma =_{\mathsf{R}} L_i\varphi_{i-1}$ for all $1 \le i \le n$ then $\varphi_n \vdash^{R\sigma} t\sigma$
$(T,\varphi_0,\sigma) \models \mathsf{i}_{\ell_1,\ldots,\ell_i}(R,R')$	if there exists t s.t. $(T,\varphi_0,\sigma) \models \mathsf{k}_{\ell_1,\ldots,\ell_i}(R,t)$ and $(T,\varphi_0,\sigma) \models \mathsf{k}_{\ell_1,\ldots,\ell_i}(R',t)$
$(T,\varphi_0,\sigma) \models \mathsf{ri}_{\ell_1,\ldots,\ell_i}(R,R')$	if $(T,\varphi_0,\sigma) \models \mathsf{r}_{\ell_1,\ldots,\ell_i}$ and $(T,\varphi_0,\sigma) \models \mathsf{i}_{\ell_1,\ldots,\ell_i}(R,R')$

Fig. 1: Predicates

Let $T = a_1.a_2.\ldots.a_n$ be a ground trace. We assume the following naming conventions: *(i)* if a_i is a receive action then $a_i = \mathbf{in}(c_i, x_i)$; *(ii)* $x_i \neq x_j$ for any $i \neq j$; *(iii)* if a_i is a send action then $a_i = \mathbf{out}(c_i, t_i)$; *(iv)* if a_i is a test action then $a_i = [s_i \stackrel{?}{=} t_i]$. Moreover, for each $1 \le i \le n$, let $\ell_i \in \mathsf{SLabels}$ be as follows:

$$\ell_i = \begin{cases} \mathbf{in}(c_i, x_i) & \text{if } a_i = \mathbf{in}(c_i, x_i) \\ \mathbf{out}(c_i) & \text{if } a_i = \mathbf{out}(c_i, t_i) \\ \mathbf{test} & \text{if } a_i = [s_i \stackrel{?}{=} t_i] \end{cases}.$$

For each $0 \le m \le n$, let the sets $R(m)$, $S(m)$ and $T(m)$ respectively denote the indices of the receive actions, send actions and test actions amongst $a_1, \ldots, a_m$. Formally, $R(m) = \{i \mid 1 \le i \le m, a_i = \mathbf{in}(c_i, x_i)\}$, $S(m) = \{i \mid 1 \le i \le m, a_i = \mathbf{out}(c_i, t_i)\}$ and $T(m) = \{i \mid 1 \le i \le m, a_i = [s_i \stackrel{?}{=} t_i]\}$ Given a set of public names $\mathcal{M}_0 \subseteq \mathcal{M}$, *set of seed statements* associated to T and $\mathcal{M}_0$, denoted $\mathsf{seed}(T, \mathcal{M}_0)$, is defined to be the set of statements given in Figure 2. If $\mathcal{M}_0 = \mathcal{M}$, then $\mathsf{seed}(T, \mathcal{M})$ is said to be the set of seed statements associated to T and in this case we write $\mathsf{seed}(T)$ as a shortcut for $\mathsf{seed}(T, \mathcal{M})$. While constructing $\mathsf{seed}(T, \mathcal{M})$, we apply $\mathsf{mgu}_{\mathsf{R}}$ to all tests. In addition, we also apply finite variants. This allows us to *get rid* of rewriting in our procedure.

For a set of statements K, we denote by $\mathcal{H}(K)$ the least Herbrand model of $K \cup \{\mathsf{k}_{\ell_1,\ldots,\ell_{n+1}}(X,x) \Leftarrow \mathsf{k}_{\ell_1,\ldots,\ell_n}(X,x)\}_{n\in\mathbb{N}} \cup \{\mathsf{i}_{\ell_1,\ldots,\ell_{n+1}}(X_1,X_2) \Leftarrow \mathsf{i}_{\ell_1,\ldots,\ell_n}(X_1,X_2)\}_{n\in\mathbb{N}}$. We show that as far as reachability predicates and intruder knowledge predicates are concerned, the set $\mathsf{seed}(T)$ is a complete abstraction T.

$\mathsf{r}_{\ell_1\sigma\tau\downarrow,\ldots,\ell_m\sigma\tau\downarrow} \Leftarrow \{\mathsf{k}_{\ell_1\sigma\tau\downarrow,\ldots,\ell_{j-1}\sigma\tau\downarrow}(X_j, x_j\sigma\tau\downarrow)\}_{j\in R(m)}$
for all $0 \leq m \leq n$
for all $\sigma \in \mathrm{mgu}_{\mathsf{R}}(\{s_k = t_k\}_{k\in T(m)})$
for all $\tau \in \mathrm{variants}(\ell_1\sigma, \ldots, \ell_m\sigma)$

$\mathsf{k}_{\ell_1\tau\downarrow,\ldots,\ell_m\tau\downarrow}(w_{|S(m)|}, t_m\tau\downarrow) \Leftarrow \{\mathsf{k}_{\ell_1\tau\downarrow,\ldots,\ell_{j-1}\tau\downarrow}(X_j, x_j\tau\downarrow)\}_{j\in R(m)}$
for all $m \in S(n)$
for all $\tau \in \mathrm{variants}(\ell_1, \ldots, \ell_m, t_m)$

$\mathsf{k}(c, c) \Leftarrow$
for all public names $c \in \mathcal{M}_0$

$\mathsf{k}_{\ell_1,\ldots,\ell_m}(f(Y_1, \ldots, Y_k), f(y_1, \ldots, y_k)\tau\downarrow) \Leftarrow \{\mathsf{k}_{\ell_1,\ldots,\ell_m}(Y_j, y_j\tau\downarrow)\}_{j\in\{1,\ldots,k\}}$
for all $0 \leq m \leq n$
for all function symbols f of arity k
for all $\tau \in \mathrm{variants}(f(y_1, \ldots, y_k))$.

Fig. 2: Seed statements

Theorem 2. *Let T be a ground trace.*

- *(Soundness.) For any $f \in \mathsf{seed}(T) \cup \mathcal{H}(\mathsf{seed}(T))$ we have that $T \models f$.*
- *(Completeness.) If $(T, \emptyset) \xrightarrow{L_1,\ldots,L_m} (S, \varphi)$ then (i) $\mathsf{r}_{L_1\varphi\downarrow,\ldots,L_m\varphi\downarrow} \in \mathcal{H}(\mathsf{seed}(T))$, and (ii) if $\varphi \vdash^R t$ then $\mathsf{k}_{L_1\varphi\downarrow,\ldots,L_m\varphi\downarrow}(R, t\downarrow) \in \mathcal{H}(\mathsf{seed}(T))$.*

Remark 2. Note that the set $\mathsf{seed}(T)$ is only partially complete as we have not shown above that if $\varphi \vdash^R t$ and $\varphi \vdash^{R'} t$ then $\mathsf{i}_{L_1\varphi\downarrow,\ldots,L_m\varphi\downarrow} \in \mathcal{H}(\mathsf{seed}(T))$. We will shortly show how the completeness of $\mathsf{seed}(T)$ can be built upon to achieve a) full abstraction of T and b) procedures for checking equivalences $\approx_{ct}$ and $\sqsubseteq_{ft}$.

5 Procedure for deciding trace equivalence

We now present a procedure for verifying trace equivalence. At a high level, this consists of the following two steps that we will detail later.

1. A saturation procedure which constructs a set of *simple* statements from the set $\mathsf{seed}(T)$ which we will call *solved* statements. The saturation procedure ensures that the set of solved statements is a complete abstraction of T.
2. Given two ground processes P and Q, we saturate the set of seed statements for traces of P and Q and then use the solved statements to decide whether P and Q are trace equivalent.

5.1 Knowledge bases and saturation

The saturation procedure manipulates a set of statements called a knowledge base.

$$\textsc{Resolution}\ \frac{\begin{array}{c} f \in K, g \in K_{\mathsf{solved}}, \\ f = \Big(H \Leftarrow \mathsf{k}_{uv}(X,t), B_1, \ldots, B_n\Big) \qquad g = \Big(\mathsf{k}_w(R,t') \Leftarrow B_{n+1}, \ldots, B_m\Big) \\ \sigma = \mathrm{mgu}(\mathsf{k}_u(X,t), \mathsf{k}_w(R,t')) \qquad t \notin \mathcal{X} \end{array}}{K = K \oplus h \text{ where } h = \Big((H \Leftarrow B_1, \ldots, B_m)\sigma\Big)}$$

$$\textsc{Equation}\ \frac{\begin{array}{c} f, g \in K_{\mathsf{solved}}, \qquad f = \Big(\mathsf{k}_u(R,t) \Leftarrow B_1, \ldots, B_n\Big) \\ g = \Big(\mathsf{k}_{u'v'}(R',t') \Leftarrow B_{n+1}, \ldots, B_m\Big) \qquad \sigma = \mathrm{mgu}(\mathsf{k}_u(\_,t), \mathsf{k}_{u'}(\_,t')) \end{array}}{K = K \oplus h \text{ where } h = \Big((\mathsf{i}_{u'v'}(R,R') \Leftarrow B_1, \ldots, B_m)\sigma\Big)}$$

$$\textsc{Test}\ \frac{\begin{array}{c} f, g \in K_{\mathsf{solved}}, \qquad f = \Big(\mathsf{i}_u(R,R') \Leftarrow B_1, \ldots, B_n\Big) \\ g = \Big(\mathsf{r}_{u'v'} \Leftarrow B_{n+1}, \ldots, B_m\Big) \qquad \sigma = \mathrm{mgu}(u,u') \end{array}}{K = K \oplus h \text{ where } h = \Big((\mathsf{ri}_{u'v'}(R,R') \Leftarrow B_1, \ldots, B_m)\sigma\Big)}$$

Fig. 3: Saturation rules

Definition 10. *Given a statement* $f = H \Leftarrow B_1, \ldots, B_n$,

- f *is said to be* solved *if for all* $1 \leq i \leq n$, $B_i = \mathsf{k}_{\ell_1,\ldots,\ell_{j_i}}(X_i, x_i)$ *for some variables* $x_i \in \mathcal{X}, X_i \in \mathcal{Y}$.
- f *is said to be* well-formed *if whenever it is solved and* $H = \mathsf{k}_{\ell_1,\ldots,\ell_k}(R,t)$, *we have that* $t \notin \mathcal{X}$.

A set of well-formed *statements is called a* knowledge base. *If* K *is a knowledge base, we define* $K_{\mathsf{solved}} = \{f \in K \mid f \text{ is solved}\}$ *to be the knowledge base restricted to the solved statements.*

Given an initial knowledge base K, the saturation procedure produces another knowledge base $\mathsf{sat}(K)$ as follows. First, new statements are *generated*. Then the knowledge base is *updated* with the new statements. This two-step process continues until a fixed-point is achieved. We describe the two steps in the procedure.

Generating new statements. Given a knowledge base K, new statements f are generated by applying the rules in Figure 3.

Update. The first step while updating the knowledge base by f is to convert f into a canonical form.

Definition 11. *Given a solved deduction statement* f, *we define its* canonical form *to be the statement* $f\Downarrow$ *obtained by first applying Rule* Rename *as many times as possible and then applying Rule* Remove *as many times as possible:*

$$\textsc{Rename}\ \frac{H \Leftarrow \mathsf{k}_u(X,x), \mathsf{k}_{uv}(Y,x), B_1, \ldots, B_n}{(H \Leftarrow \mathsf{k}_u(X,x), B_1, \ldots, B_n)\{Y \mapsto X\}}$$

$$\textsc{Remove}\ \frac{H \Leftarrow \mathsf{k}_u(X,x), B_1, \ldots, B_n \qquad x \notin \mathit{vars}(H)}{H \Leftarrow B_1, \ldots, B_n}$$

For any other type of statement, the canonical form $f\Downarrow$ is defined to be f.

It is easy to see that any fact f can be converted into a canonical form. After a canonical form has been obtained, we perform another check before $f\Downarrow$ can be added to the knowledge base. Intuitively, this check ensures that we add enough identity predicates in the knowledge base. We need the following definition for the update rule.

Definition 12. *The set of* consequences *of a knowledge base K, denoted* $\mathbf{cons}(K)$*, is the smallest set such that:*

$$\textsc{Axiom}\ \frac{}{\mathsf{k}_{uv}(R,t) \Leftarrow \mathsf{k}_u(R,t), B_1, \ldots, B_m \in \mathbf{cons}(K)}$$

$$\textsc{Res}\ \frac{H \Leftarrow B_1, \ldots, B_n \in K \qquad \sigma \text{ a substitution} \qquad B_1\sigma \Leftarrow C_1, \ldots, C_m \in \mathbf{cons}(K), \ldots, B_n\sigma \Leftarrow C_1, \ldots, C_m \in \mathbf{cons}(K)}{H\sigma \Leftarrow C_1, \ldots, C_m \in \mathbf{cons}(K)}$$

Given a knowledge base K and a statement f, the *update of K by f*, denoted $K \oplus f$, is defined to be $K \cup \{f\Downarrow\}$ if the head of f is not of the form $\mathsf{k}_{\ell_1,\ldots,\ell_k}(R,t)$. Otherwise, let

$$f\Downarrow = \mathsf{k}_{\ell_1,\ldots,\ell_k}(R,t) \Leftarrow \mathsf{k}_{\ell_1,\ldots,\ell_{i_1}}(X_1,t_1), \ldots, \mathsf{k}_{\ell_1,\ldots,\ell_{i_n}}(X_n,t_n)$$

and $K \oplus f =$

- $K \cup \{f\Downarrow\}$ if f is solved and for any R' we have that $\mathsf{k}_{\ell_1,\ldots,\ell_k}(R',t) \Leftarrow \mathsf{k}_{\ell_1,\ldots,\ell_{i_1}}(X_1,t_1), \ldots, \mathsf{k}_{\ell_1,\ldots,\ell_{i_n}}(X_n,t_n) \notin \mathbf{cons}(K_{\mathsf{solved}})$.
- $K \cup \{\mathsf{i}_{\ell_1,\ldots,\ell_k}(R,R') \Leftarrow \{\mathsf{k}_{\ell_1,\ldots,\ell_{i_j}}(X_j,t_j)\}_{j\in\{1,\ldots,n\}}\}$ if f is solved and R' is such that $\mathsf{k}_{\ell_1,\ldots,\ell_k}(R',t) \Leftarrow \mathsf{k}_{\ell_1,\ldots,\ell_{i_1}}(X_1,t_1), \ldots, \mathsf{k}_{\ell_1,\ldots,\ell_{i_n}}(X_n,t_n) \in \mathbf{cons}(K_{\mathsf{solved}})$.
- $K \cup \{f\Downarrow\}$ if f is not solved.

Note that update is not a function, namely that there may be several $R', i_1, \ldots, i_n$ such that $\mathsf{k}_{\ell_1,\ldots,\ell_k}(R',t) \Leftarrow \mathsf{k}_{\ell_1,\ldots,\ell_{i_1}}(X_1,t_1), \ldots, \mathsf{k}_{\ell_1,\ldots,\ell_{i_n}}(X_n,t_n) \in \mathbf{cons}(K_{\mathsf{solved}})$. However, we need to compute only one such R'.

Initial knowledge base. One question that naturally arises is what is the initial knowledge base for the saturation procedure. Given a ground trace T, the initial knowledge base for the saturation procedure is defined as follows.

Definition 13. *Given a set of statements S, the* initial knowledge base *associated to S, denoted $K_i(S)$, is defined to be the empty knowledge base updated by the set S, i.e., $K_i(S) = \emptyset \oplus_{f\in S} f$. If T is a ground trace, we write $K_i(T)$ for $K_i(\mathsf{seed}(T))$.*

Observe that $K_i(T)$ depends on the order in which statements in $\mathsf{seed}(T)$ are updated. The exact order, however, is not important and our results hold regardless of the order chosen. The saturation procedure takes $K_i(T)$ as an input and

produces a knowledge base $\mathsf{sat}(K_i(T))$. The reason for choosing $K_i(T)$ instead of $\mathsf{seed}(T)$ as the starting point of the saturation procedure is that $\mathsf{seed}(T)$ may not be a knowledge base, i.e., may contain non well-formed statements. The set $K_i(T)$ is, however, a knowledge base.

Proposition 1. *Given a ground trace T, the set $K_i(T)$ is a knowledge base.*

Soundness and completeness of the saturation procedure. We shall now show that the set of solved statements in $\mathsf{sat}(K_i(T))$ is a sound and complete abstraction of a ground trace T. Given a set of statements K we denote by $\mathcal{H}_{\mathsf{e}}(K)$ the smallest set of ground terms such that

- $\mathcal{H}(K) \subseteq \mathcal{H}_{\mathsf{e}}(K)$,
- $\mathcal{H}_{\mathsf{e}}(K)$ is closed under congruence rules for each $\mathsf{i}_w(R, R') \in \mathcal{H}_{\mathsf{e}}(K)$, and
- i_w is monotonic in w, i.e., $\mathsf{i}_u(R, R') \in \mathcal{H}_{\mathsf{e}}(K)$ implies $\mathsf{i}_{uv}(R, R') \in \mathcal{H}_{\mathsf{e}}(K)$.

A formal definition is given in [16].

Theorem 3. *Let T be a ground trace and let $K = \mathsf{sat}(K_i(T))$.*

- *(Soundness.) For any $f \in K \cup \mathcal{H}_{\mathsf{e}}(K)$ we have $T \models f$.*
- *(Completeness.) If $(T, \emptyset) \xrightarrow{L_1,\ldots,L_n} (S, \varphi)$ then (i) $\mathsf{r}_{L_1\varphi\downarrow,\ldots,L_n\varphi\downarrow} \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})$, (ii) if $\varphi \vdash^R t$ then $\mathsf{k}_{L_1\varphi\downarrow,\ldots,L_n\varphi\downarrow}(R, t\downarrow) \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})$, and (iii) if $\varphi \vdash^R t$ and $\varphi \vdash^{R'} t$, then $\mathsf{i}_{L_1\varphi\downarrow,\ldots,L_n\varphi\downarrow}(R, R') \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})$.*

Effectiveness of the saturation procedure. We have shown that the set of solved statements in $\mathsf{sat}(K_i(T))$ form a sound and complete abstraction for the trace T. However this set is infinite and may not be effectively computable. This may be because of following reasons.

- The set $\mathsf{seed}(T)$ for a ground trace T is infinite. Hence the saturation procedure may continue forever. We will, however, shortly show that for the saturation procedure we only need to consider the saturation of the set $K_i(\mathsf{seed}(T, \mathcal{M}_0))$ where $\mathcal{M}_0$ is the set of public names occurring in T (see Lemma 1). The set $\mathsf{sat}(K_i(T))$ can then be computed from this set. Since the set $K_i(\mathsf{seed}(T, \mathcal{M}_0))$ is finite, this means that all intermediate knowledge bases in the saturation procedure are finite.
- For the update rule, we have to check that given a knowledge base K, term t, labels $\ell_1, \ldots, \ell_k$, indices $1 \leq i_1, \ldots i_n \leq k$, variables $x_1, \ldots, x_n \in \mathcal{X}$ and recipe variables $X_1, \ldots, X_n \in \mathcal{Y}$, whether

  $$\exists R.\ \mathsf{k}_{\ell_1,\ldots,\ell_k}(R, t) \Leftarrow \mathsf{k}_{\ell_1,\ldots,\ell_{i_1}}(X_1, x_1), \ldots, \mathsf{k}_{\ell_1,\ldots,\ell_{i_n}}(X_n, x_n) \in \mathbf{cons}(K_{\mathsf{solved}}).$$

  Furthermore, if the check succeeds then we have to compute one such R. We will show that can be achieved if K is finite (see Lemma 2).
- The saturation procedure may itself not terminate even if the initial knowledge base is finite. As pointed out in the Introduction, we conjecture that the saturation procedure terminates for subterm convergent rewrite systems, but were unable to show the termination.

The following lemma allows us to compute the $\mathsf{sat}(K_i(T))$ from the set $\mathsf{sat}(K_i(seed(M_0, T)))$ where M_0 is the set of public names occurring in T.

Lemma 1. *Let T be a ground trace and $M_T \subseteq \mathcal{M}$ be the public names occurring in T. Let $K_{\mathcal{M}} = \{\{\mathsf{k}(m,m) \Leftarrow\}_{m\in\mathcal{M}} \cup \{\mathsf{i}(m,m) \Leftarrow\}_{m\in\mathcal{M}} \cup \{\mathsf{ri}(m,m) \Leftarrow \}_{m\in\mathcal{M}}\}$. Then $\mathsf{sat}(K_i(T)) = \mathsf{sat}(K_i(seed(M_T, T))) \cup K_{\mathcal{M}}$.*

The following lemma implies that the update step terminates if we only have a finite number of solved statements in the knowledge base.

Lemma 2. *Given a finite set of statements K, term t, labels $\ell_1, \ldots, \ell_k$, indices $1 \leq i_1, \ldots i_n \leq k$, variables $x_1, \ldots, x_n \in \mathcal{X}$ and recipe variables $X_1, \ldots, X_n \in \mathcal{Y}$, it is decidable if there is an R such that $\mathsf{k}_{\ell_1,\ldots,\ell_k}(R, t) \Leftarrow \mathsf{k}_{\ell_1,\ldots,\ell_{i_1}}(X_1, x_1)$, $\ldots, \mathsf{k}_{\ell_1,\ldots,\ell_{i_n}}(X_n, x_n) \in \mathbf{cons}(K_{\mathsf{solved}})$. If the answer to the decision procedure is "Yes", then we can compute one such R.*

5.2 Algorithm for checking equivalence

Once we constructed saturated knowledge bases for the seed statements for ground determinate processes P_0 and P_1, we can check trace equivalence $\approx_{ct}$. The algorithm for checking $\approx_{ct}$ for determinate processes, automatically gives an algorithm for checking $\approx_{ft}$ for non-determinate processes. It suffices to check for $T \sqsubseteq_{ct} P$ for a ground trace T and ground determinate process P. This basically involves checking two tests which are summarized in Figure 4. We briefly describe them below.

- Reach checks whether all sequence of actions executable by T are also executable by P. To do this, we carry out the following operations for *each* statement $\mathsf{r}_{l_1,\ldots,l_n} \Leftarrow \{\mathsf{k}_{w_i}(X_i, x_i)\}_{i\in\{1,\ldots,m\}}\Big) \in \{\mathsf{sat}(\mathsf{seed}(T))\}_{\mathsf{solved}}$. *(a)* First we pick fresh constants $c_1, \ldots, c_k$ for each of the variables occurring in $l_1, \ldots, l_n$ and fix a bijection σ between them. *(b)* Next for each $1 \leq i \leq n$ s.t. l_i is $\mathbf{in}(d_i, t_i)$, we construct *one* recipe R_i such that $\mathsf{k}_{l_1\sigma,\ldots,l_{i-1}\sigma}(R_i, t_i\sigma) \in \mathcal{H}(\{\mathsf{sat}(\mathsf{seed}(T))\}_{\mathsf{solved}})$. Such an R_i exists thanks to the completeness of the saturation procedure. We let $M_i = \mathbf{in}(d_i, R_i)$. *(c)* For each $1 \leq i \leq n$ s.t. $l_i = \mathbf{test}$ or $\mathbf{out}(d_i)$ we let $M_i = l_i$. *(d)* We check if $(P, \emptyset) \xRightarrow{M_1,\ldots,M_n} (T', \varphi)$. If all the Reach tests pass then we go to test Identity. Otherwise we declare T to be not trace-contained in P.
- The test Identity checks that all the equality tests that hold after an execution of T hold after a similar execution in P. In order to do this, we carry out the following operations for *each* statement $\mathsf{ri}_{l_1,\ldots,l_n}(R, R') \Leftarrow \{\mathsf{k}_{w_i}(X_i, x_i)\}_{i\in\{1,\ldots,m\}}\Big) \in \{\mathsf{sat}(\mathsf{seed}(T))\}_{\mathsf{solved}}$. We construct $M_1, \ldots, M_n$ as in the Reach test and check if there is a T' such that $(P, \emptyset) \xRightarrow{M_1,\ldots,M_n} (T', \varphi)$ and the recipes $R\{ X_i \mapsto x_i\sigma \}$ and $R'\{ X_i \mapsto x_i\sigma \}$ are equal in frame φ.

Note that performing the tests requires deciding if, given t, and w, $\mathsf{k}_w(R, t) \in \mathcal{H}(K)$ for some recipe R for a knowledge base K containing only solved statements. This is similar to checking if $\Big(\mathsf{k}_w(R, t) \Leftarrow\Big) \in \mathbf{cons}(K)$.

Theorem 4. *Let T be a ground trace and let P be a ground determinate process. Let K be the set of solved statements from a saturated knowledge base associated to T. Then $T \sqsubseteq_{ct} P$ iff all the tests in Figure 4 hold.*

$$\textsc{Reach}\ \frac{\begin{array}{c}\left(\mathsf{r}_{l_1,\ldots,l_n} \Leftarrow \{\mathsf{k}_{w_i}(X_i, x_i)\}_{i\in\{1,\ldots,m\}}\right) \in \{\mathsf{sat}(\mathsf{seed}(T))\}_{\mathsf{solved}} \\ c_1,\ldots,c_k \text{ fresh constants} \\ \sigma : vars(l_1,\ldots,l_n) \to \{c_1,\ldots,c_k\} \text{ is a bijection} \\ \mathsf{k}_{l_1\sigma,\ldots,l_{i-1}\sigma}(R_i, t_i\sigma) \in \mathcal{H}(\{\mathsf{sat}(\mathsf{seed}(T))\}_{\mathsf{solved}}) \text{ for all } i \text{ s.t. } l_i = \mathbf{in}(d_i, t_i) \\ M_i = l_i \text{ if } l_i \in \{\mathbf{test}, \mathbf{out}(\_)\} \qquad M_i = \mathbf{in}(d_i, R_i) \text{ if } l_i = \mathbf{in}(d_i, t_i)\end{array}}{(P,\emptyset) \xrightarrow{M_1,\ldots,M_n} (T',\varphi)}$$

$$\textsc{Identity}\ \frac{\begin{array}{c}\left(\mathsf{ri}_{l_1,\ldots,l_n}(R, R') \Leftarrow \{\mathsf{k}_{w_i}(X_i, x_i)\}_{i\in\{1,\ldots,m\}}\right) \in \{\mathsf{sat}(\mathsf{seed}(T))\}_{\mathsf{solved}} \\ c_1,\ldots,c_k \text{ fresh constants} \\ \sigma : vars(l_1,\ldots,l_n) \to \{c_1,\ldots,c_k\} \text{ is a bijection} \\ \mathsf{k}_{l_1\sigma,\ldots,l_{i-1}\sigma}(R_i, t_i\sigma) \in \mathcal{H}(\{\mathsf{sat}(\mathsf{seed}(T))\}_{\mathsf{solved}}) \text{ for all } i \text{ s.t. } l_i = \mathbf{in}(t_i) \\ M_i = l_i \text{ if } l_i \in \{\mathbf{test}, \mathbf{out}(\_)\} \qquad M_i = \mathbf{in}(d_i, R_i) \text{ if } l_i = \mathbf{in}(d_i, t_i)\end{array}}{(P,\emptyset) \xrightarrow{M_1,\ldots,M_n} (T',\varphi) \text{ such that } (R\omega = R'\omega)\varphi \text{ where } \omega = \{X_i \mapsto x_i\sigma\}}$$

Fig. 4: Tests for checking trace inclusion

6 Prototype and case studies

We implemented the procedure for checking equivalence in a prototype, AKiSs (Active Knowledge in Security protocols). AKiSs is written in OCaml and has about 2000 lines of source code, including code for computing complete sets of finite variants and complete sets of equational unifiers. For protocol specification, we allow for an operator *interleave* which models parallel composition of processes and an operator *sequence* for modeling protocols structured in phases.

We used AKiSs to verify the equivalences in Examples 5 and 6. Using AKiSs we were able to verify strong secrecy for Denning-Sacco-Blanchet [11] and Needham-Schroeder-Lowe (NSL) [36], resistance to guessing attacks in the EKE protocol [9], and, more interestingly, anonymity of the FOO [32] and Okamoto [38] electronic voting protocols.[3] To our knowledge, AKiSs is the only tool that can verify FOO and Okamoto automatically. We briefly discuss the salient points of these examples below. AKiSs along with all the discussed examples is available on: `http://www.lsv.ens-cachan.fr/~ciobaca/akiss/`. Details of the modeling can also be found in [16].

[3] Please note that as defined in [38], modeling of Okamoto's protocol requires private channels. As we do not have private channels in our calculus, we transform the protocol so that every message sent by honest participants on a private channel is sent encrypted under a key not known to the adversary

Strong flavors of confidentiality. The *strong secrecy* property was introduced by Blanchet in [11] and we rephrase it here in our setting. Let P be a protocol with x as the only free variable of P. Then x is said to be *strongly secret* if

$$\mathbf{in}(c, x_1).\mathbf{in}(c, x_2).(P\{x \mapsto x_1\}) \approx_t \mathbf{in}(c, x_1).\mathbf{in}(c, x_2).(P\{x \mapsto x_2\}).$$

Intuitively, the attacker cannot distinguish the processes using variables x_1 and x_2 even though it can choose arbitrary (public) values for these variables. The definition generalizes to multiple variables in the expected way. We illustrate this property on a Denning-Sacco-Blanchet protocol. Informally, the protocol can be described as follows.

$$\mathsf{A} \to \mathsf{B} : \mathsf{aenc}(\mathsf{sign}(\mathsf{pair}(\mathsf{pk}(ska), \mathsf{pair}(\mathsf{pk}(skb, k))), ska), \mathsf{pk}(skb))$$
$$\mathsf{B} \to \mathsf{A} : \mathsf{enc}(x, k)$$

A sends to B a fresh symmetric session key k together with A's and B's public keys. This is signed with A's secret key and (asymmetrically) encrypted with B's public key. Upon receiving this message, B decrypts it, checks the signature and uses the fresh session key to symmetrically encrypt a secret x. We used AKISs to verify this protocol for strong secrecy of x (with one session of A and B). This protocol is determinate, and hence we used $\approx_{ct}$ to verify the protocol. The verification succeeds as expected.

A variant of the protocol [11] consists in letting A also send out a secret y encrypted with k changing the first message to

$$\mathsf{A} \to \mathsf{B} : \mathsf{pair}(\mathsf{aenc}(\mathsf{sign}(\mathsf{pair}(\mathsf{pk}(ska), \mathsf{pair}(\mathsf{pk}(skb, k))), ska), \mathsf{pk}(skb)), \mathsf{enc}(y, k))$$

In this case the protocol does not respect strong secrecy of x, y as, by choosing $x_1 = y_1$ and $x_2 \neq y_2$, the attacker can distinguish the two situations by testing the equality of the encryptions of x and y. This attack is again found by AKISs. AKISs also verifies strong secrecy of the nonce generated by the responder in the Needham-Schroeder-Lowe (NSL) [36] protocol. Once again, the modeling of NSL leads to determinate processes, and we used $\approx_{ct}$ for our verification.

We also used AKISs to verify the above protocols for *real-or-random* secrecy. This property is useful to model resistance to offline guessing attacks in password protocols [8]. We show that the EKE protocol [9] is resistant to offline guessing attacks. As EKE also leads to determinate processes, we used the $\approx_{ct}$ relation.

Anonymity for electronic voting protocol. A voting protocol must respect voter privacy: the adversary should not be able to learn how each voter voted. AKISs can automatically verify voter privacy in the FOO electronic voting protocol [32] and the Okamoto protocol [38]. Voter privacy is naturally modeled as an equivalence property [26, 7]: it is not possible to distinguish the situation where honest voter A votes 'yes' and honest B votes 'no' from the situation that A votes 'no' and B votes 'yes'. Note that our modeling of the protocols is exactly the same as in [26]. We assume that *only* voters A and B are honest while all other entities are dishonest. An arbitrary number of dishonest voters are however subsumed

by the attacker and need not be modeled directly. Both the protocols do not lead to determinate processes. Therefore, we proved the relation $\approx_{ft}$. To our knowledge, no other tool can handle this automatically. We are aware of two other attempts for verifying the FOO protocol. Using ProVerif [11], Delaune *et al.* [28], verify a transformation of the protocol. However, the soundness of this transformation has never been proven. Chothia *et al.* [20] verify a different notion of anonymity (also based on process equivalence) using the μCRL tool. However, the attacker they consider is only an observer that cannot interact with the protocol participants, yielding a finite state system.

Efficiency. On a standard modern laptop, AKiSs takes a few minutes (e.g. 3 mins for FOO) to carry out the above verification. The use of a multi-core server already reduces these timings by about 40%. We expect that some optimizations of the saturation procedure and the use of more efficient data structures will diminish these times significantly. Most of the computational effort goes into the saturation of the traces. Interleaving individual roles of a protocol introduces an exponential blowup on the number of traces and saturations to perform. However, it would be straightforward to scale to larger protocols and more sessions by parallelizing the saturation of these traces (e.g. on clusters of machines).

7 Conclusion and future work

We present a novel Horn-clause resolution based procedure for verifying equivalence properties for a bounded number of sessions of cryptographic protocols. This approach is validated by implementing it in the tool AKiSs, and we are able to handle examples which are out of the scope of existing tools.

There are several directions for future work. The implementation of the tool should be optimized and more examples from electronic voting, RFID protocols and auction protocols which all have requirements stated in terms of equivalences should be analyzed. We would also like to take disequalities into account. It will allow to verify processes with else branches, important in a number of practical examples, e.g., passport protocols discussed in [5]. Another direction would be to extend the procedure to allow AC (Associative/Commutative) operators in order to treat protocols based on exclusive-or or Diffie-Hellman exponentiations.

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 *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. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Inf. Comput.*, 148(1):1–70, 1999.

5. M. Arapinis, T. Chothia, E. Ritter, and M. D. Ryan. Analysing unlinkability and anonymity using the applied pi calculus. In *23rd Computer Security Foundations Symposium (CSF'10)*, pages 107–121. IEEE Comp. Soc. Press, 2010.
6. A. Armando et al. The AVISPA tool for the automated validation of internet security protocols and applications. In *17th International Conference on Computer Aided Verification (CAV'05)*, LNCS, pages 281–285. Springer, 2005.
7. M. Backes, C. Hritcu, and M. Maffei. Automated verification of remote electronic voting protocols in the applied pi-calculus. In *21st Computer Security Foundations Symposium (CSF'08)*. IEEE Comp. Soc. Press, 2008.
8. M. Baudet. Deciding security of protocols against off-line guessing attacks. In *12th Conference on Computer and Communications Security (CCS'05)*, pages 16–25. ACM Press, 2005.
9. S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Symposium on Security and Privacy (S&P'92)*, pages 72–84. IEEE Comp. Soc. Press, 1992.
10. B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *14th Computer Security Foundations Workshop (CSFW'01)*, pages 82–96. IEEE Comp. Soc. Press, 2001.
11. B. Blanchet. Automatic proof of strong secrecy for security protocols. In *Symposium on Security and Privacy (S&P'04)*, pages 86–100, 2004.
12. B. Blanchet, M. Abadi, and C. Fournet. Automated Verification of Selected Equivalences for Security Protocols. In *Symposium on Logic in Computer Science*, pages 331–340. IEEE Comp. Soc. Press, 2005.
13. J. Borgström. *Equivalences and Calculi for Formal Verifiation of Cryptographic Protocols*. Phd thesis, EPFL, Switzerland, 2008.
14. J. Borgström, S. Briais, and U. Nestmann. Symbolic bisimulation in the spi calculus. In *15th Int. Conference on Concurrency Theory (CONCUR'04)*, volume 3170 of *LNCS*, pages 161–176. Springer, 2004.
15. M. Bruso, K. Chatzikokolakis, and J. den Hartog. Analysing unlinkability and anonymity using the applied pi calculus. In *23rd Computer Security Foundations Symposium (CSF'10)*, pages 107–121. IEEE Comp. Soc. Press, 2010.
16. R. Chadha, Ş. Ciobâcă, and S. Kremer. Automated verification of equivalence properties of cryptographic protocols. Technical report, Oct. 2011. `http://hal.inria.fr/inria-00632564/en/`.
17. V. Cheval, H. Comon-Lundh, and S. Delaune. Automating security analysis: symbolic equivalence of constraint systems. In *International Joint Conference on Automated Reasoning (IJCAR'10)*, LNAI, pages 412–426. Springer, 2010.
18. V. Cheval, H. Comon-Lundh, and S. Delaune. Trace equivalence decision: Negative tests and non-determinism. In *18th Conference on Computer and Communications Security (CCS'11)*, pages 321–330. ACM Press, 2011.
19. Y. Chevalier and M. Rusinowitch. Decidability of equivalence of symbolic derivations. *Journal of Automated Reasoning*, 2010. To appear.
20. T. Chothia, S. Orzan, J. Pang, and M. Torabi Dashti. A framework for automatically checking anonymity with *mu* crl. In *2nd Symposium on Trustworthy Global Computing (TGC'06)*, volume 4661 of *LNCS*, pages 301–318. Springer, 2007.
21. Ş. Ciobâcă. Computing finite variants for subterm convergent rewrite systems. Research Report LSV-11-06, LSV, ENS Cachan, France, 2011.
22. H. Comon-Lundh and S. Delaune. The finite variant property: How to get rid of some algebraic properties. In *16th International Conference on Rewriting Techniques and Applications (RTA'05)*, volume 3467 of *LNCS*, pages 294–307. Springer, 2005.

23. V. Cortier and S. Delaune. A method for proving observational equivalence. In *22nd Computer Security Foundations Symposium (CSF'09)*, pages 266–276. IEEE Comp. Soc. Press, 2009.
24. M. Dahl, S. Delaune, and G. Steel. Formal analysis of privacy for vehicular mix-zones. In *15th European Symposium on Research in Computer Security (ESORICS'10)*, volume 6345 of *LNCS*, pages 55–70. Springer, 2010.
25. S. Delaune, S. Kremer, and O. Pereira. Simulation based security in the applied pi calculus. In *29th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'09)*, volume 4 of *Leibniz International Proceedings in Informatics*, pages 169–180. Leibniz-Zentrum für Informatik, 2009.
26. S. Delaune, S. Kremer, and M. D. Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17(4):435–487, 2009.
27. S. Delaune, S. Kremer, and M. D. Ryan. Symbolic bisimulation for the applied pi calculus. *Journal of Computer Security*, 18(2):317–377, Mar. 2010.
28. S. Delaune, M. D. Ryan, and B. Smyth. Automatic verification of privacy properties in the applied pi-calculus. In *2nd Joint iTrust and PST Conferences on Privacy, Trust Management and Security (IFIPTM'08)*, volume 263 of *IFIP Conference Proceedings*, pages 263–278. Springer, 2008.
29. D. Dolev and A. Yao. On the security of public key protocols. In *22nd Symposium on Foundations of Computer Science (FOCS'81)*, pages 350–357. IEEE Comp. Soc. Press, 1981.
30. 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.
31. S. Escobar, C. Meadows, and J. Meseguer. Maude-NPA: Cryptographic protocol analysis modulo equational properties. In *Foundations of Security Analysis and Design V*, volume 5705 of *LNCS*, pages 1–50. Springer, 2009.
32. A. Fujioka, T. Okamoto, and K. Ohta. A practical secret voting scheme for large scale elections. In *Advances in Cryptology — AUSCRYPT '92*, volume 718 of *LNCS*, pages 244–251. Springer, 1992.
33. J. Goubault-Larrecq. Deciding $\mathcal{H}_1$ by resolution. *Information Processing Letters*, 95(3):401–408, 2005.
34. H. Hüttel. Deciding framed bisimilarity. In *4th International Workshop on Verification of Infinite-State Systems (INFINITY'02)*, pages 1–20, 2002.
35. J. Liu and H. Lin. A complete symbolic bisimulation for full applied pi calculus. In *36th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'10)*, volume 5901 of *LNCS*, pages 552–563. Springer, 2010.
36. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, volume 1055 of *LNCS*, pages 147–166. Springer, 1996.
37. P. Narendran, F. Pfenning, and R. Statman. On the unification problem for cartesian closed categories. *J. Symb. Log.*, 62(2):636–647, 1997.
38. T. Okamoto. Receipt-free electronic voting schemes for large scale elections. In *5th Int. Security Protocols Workshop*, volume 1361 of *LNCS*, pages 25–35. Springer, 1997.
39. A. Tiu and J. Dawson. Automating open bisimulation checking for the spi-calculus. In *23rd Computer Security Foundations Symposium (CSF'10)*, pages 307–321. IEEE Comp. Soc. Press, 2010.
40. C. Weidenbach. Towards an automatic analysis of security protocols in first-order logic. In *16th International Conference on Automated Deduction (CADE'99)*, volume 1632 of *LNCS*, pages 314–328. Springer, 1999.