Automated Verification of Equivalence Properties of Cryptographic Protocols

ROHIT CHADHA, University of Missouri

VINCENT CHEVAL, University of Kent

ȘTEFAN CIOBĂCĂ, “Alexandru Ioan Cuza” University

STEVE KREMER, Inria Nancy–Grand-Est

Indistinguishability properties are essential in formal verification of cryptographic protocols. They are needed to model anonymity properties, strong versions of confidentiality, and resistance against offline guessing attacks. Indistinguishability properties can be conveniently modeled as equivalence properties. We present a novel procedure to verify equivalence properties for a bounded number of sessions of cryptographic protocols. As in the applied pi calculus, our protocol specification language is parametrized by a first-order sorted term signature and an equational theory that allows formalization of algebraic properties of cryptographic primitives. 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 whose equational theory is generated by an optimally reducing convergent rewrite system. The procedure is based on a fully abstract modelling of the traces of a bounded number of sessions of the protocols into first-order Horn clauses on which a dedicated resolution procedure is used to decide equivalence properties. We have shown that our procedure terminates for the class of subterm convergent equational theories. Moreover, the procedure has been implemented in a prototype tool Active Knowledge in Security Protocols and has been effectively tested on examples. Some of the examples were outside the scope of existing tools, including checking anonymity of an electronic voting protocol due to Okamoto.

Categories and Subject Descriptors: D.2.4 [Software Engineering]: Software/Program Verification; F.3.1 [Logics and Meanings of Programs]: Specifying and Verifying and Reasoning about Programs; F.3.2 [Logics and Meanings of Programs]: Semantics of Programming Languages

General Terms: Security, Verification

Rohit Chadha was supported in part by NSF CNS 1314338. Parts of this work were done while the first, third, and fourth authors were affiliated with LSV, CNRS, and Inria and ENS Cachan, and the second author was affiliated with LORIA, CNRS, and Inria and the Université de Lorraine.

Vincent Cheval and Steve Kremer were supported by JCJC VIP (decision ANR-11-JS02-006) and the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation program (grant agreement No 645865-SPOOC). The work of Ștefan Ciobăcă in this article is supported by the European Sectorial Operational Programme Human Resource Development (SOP HRD), and by the Romanian Government under the contract number POSDRU/159/1.5/S/137750.

Authors’ addresses: R. Chadha, 201 Engineering Building West, Department of Computer Science, University of Missouri, MO 65203, USA; email: chadhar@missouri.edu; Ș. Ciobăcă, Faculty of Computer Science, “Alexandru Ioan Cuza” University, 16, General Berthelot Street, Iași 700483, Romania; email: stefan.ciobaca@info.uaic.ro; V. Cheval, LORIA, Campus Scientifique, BP 239, 54506 Vandoeuvre-lès-Nancy Cedex, France; email: vincent.cheval@loria.fr; S. Kremer, LORIA, Campus Scientifique, BP 239, 54506 Vandoeuvre-lès-Nancy Cedex, France; email: Steve.Kremer@inria.fr.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.

© 2016 ACM 1529-3785/2016/09-ART23 \$15.00

DOI: <http://dx.doi.org/10.1145/2926715>

Additional Key Words and Phrases: Applied pi calculus, automated verification, process equivalence, security protocols

ACM Reference Format:

Rohit Chadha, Vincent Cheval, Ștefan Ciobăcă, and Steve Kremer. 2016. Automated verification of equivalence properties of cryptographic protocols. *ACM Trans. Comput. Logic* 17, 4, Article 23 (September 2016), 32 pages.

DOI: <http://dx.doi.org/10.1145/2926715>

1. INTRODUCTION

Cryptographic protocols are distributed programs that 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. For example, French citizens living abroad were allowed to vote via the Internet in the parliamentary elections in 2012 [Fre 2015]. In Estonia, Internet voting has been allowed in parliamentary elections since 2007 [Est 2015]. Internet voting was also deployed in the state elections in New South Wales, Australia, in 2015 [Halderman and Teague 2015]. 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 is of great importance and 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 back-box cryptography and unbounded computational power on the part of the adversary is made. This adversarial model is often called the Dolev-Yao model, as it is derived from the positions that Dolev and Yao took in their seminal article [Dolev and Yao 1981]. It has proved extremely successful, and there are several automated tools [Blanchet 2001; Armando et al. 2005; Cremers 2008; Escobar et al. 2009] that can automatically check trace-properties such as (weak forms of) confidentiality and authentication. While these 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 [Blanchet 2004]; resistance to guessing attacks in password based protocols [Baudet 2005]; and anonymity properties in private authentication [Abadi and Fournet 2004], electronic voting [Delaune et al. 2009b; Backes et al. 2008], vehicular networks [Dahl et al. 2010, 2011], and Radio-frequency identification (RFID) protocols [Arapinis et al. 2010; Bruso et al. 2010]. More generally, indistinguishability allows to model security by the means of ideal systems, which are correct by construction [Abadi and Gordon 1999; Delaune et al. 2009a]. Indistinguishability properties of cryptographic protocols are naturally modeled by the means of *observational* and *testing equivalences* in cryptographic extensions of process calculi, for example, the spi [Abadi and Gordon 1999] and the applied pi calculus [Abadi and Fournet 2001]. While we have good tools for automated verification of trace properties, the situation differs for indistinguishability properties. This article is an attempt to address this concern.

State of the art. Many results have been obtained in the restricted case of a pure eavesdropper, that is, a passive adversary: For *static equivalence*, many decidability results have been shown [Abadi and Cortier 2006; Cortier and Delaune 2007; Arnaud et al. 2007], and exact [Baudet et al. 2009; Ciobăcă et al. 2011] and approximate [Blanchet et al. 2005] tools exist for a variety of cryptographic primitives. In the case we consider, indistinguishability in the presence of an active adversary who can interact in

an arbitrary way with honest participants, fewer results are known. Hüttel [Hüttel 2002] showed undecidability of observational equivalence in the spi calculus, even for the finite control fragment, as well as decidability for the finite, that is, 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 [Blanchet et al. 2005; Cheval and Blanchet 2013; Santiago et al. 2014] allow us to approximate observational equivalence for an unbounded number of sessions. However, this approximation does not suffice to conclude for many applications, see, for example, Delaune et al. [2009b] and Arapinis et al. [2010]. Our approach overcomes these limitations for some applications in Delaune et al. [2009b]. We still cannot conclude for the e-passport example in Arapinis et al. [2010], albeit for a different reason: Our procedure does not currently handle else branches in protocols.

Symbolic bisimulations have also been devised for the spi [Borgström et al. 2004; Borgström 2008; Tiu and Dawson 2010] and applied pi calculus [Delaune et al. 2010; Liu and Lin 2010] to avoid unbounded branching due to adversary inputs. However, only Delaune et al. [2010], Tiu and Dawson [2010], and Borgström et al. [2004] yield a decision procedure but again only approximating observational equivalence. The results of Delaune et al. [2010] have been further refined to show a decision procedure on a restricted class of *simple* processes [Cortier and Delaune 2009]. In particular, they rely on a procedure deciding the equivalence of constraint systems, introduced by Baudet [Baudet 2005], 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 system [Abadi and Cortier 2006]. An alternate procedure achieving the same goal was proposed by Chevalier and Rusinowitch [Chevalier and Rusinowitch 2010]. However, both procedures are highly non-deterministic and do not yield a reasonable algorithm that could be implemented. Therefore, Cheval et al. [2010] 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 [Durante et al. 2003], open bisimulation [Tiu and Dawson 2010], and trace equivalence [Cheval et al. 2011] for a bounded number of sessions but only a limited set of primitives. One may note that [Cheval et al. 2011] is the only decision procedure to consider negative tests, that is, else branches, which are crucial in several case studies [Arapinis et al. 2010; Abadi and Fournet 2004].

Our contribution. In this article, we introduce a new procedure for verifying equivalence properties for processes specified in a cryptographic process calculus (without replication). The messages in the calculus are modeled as terms equipped with an equational theory, similarly to the applied pi calculus. Our main contributions are as follows.

- Our procedure checks for two equivalences that 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 have shown in Cortier and Delaune [2009] 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 the coarser relation coincides with $\approx_t$, and thus 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 for a *bounded* number of sessions in *first-order Horn clauses*. This is in contrast to the constraint-solving techniques employed by Tiu and Dawson [2010], Cheval et al. [2010], Cheval et al. [2011], Baudet [2005], and Chevalier and Rusinowitch [2010] for verifying under-approximations of observational equivalence. Techniques

based on Horn clauses have been extensively used, for example, by Blanchet [2001], Weidenbach [1999], and Goubault-Larrecq [2005], in the case of an unbounded number of sessions. Affeldt and Comon-Lundh [2009] faithfully encode a bounded protocol into Horn clauses with rigid variables. Of these tools, only Blanchet [2001] can verify an equivalence property, which happens to be an under-approximation of observational equivalence. Horn clause modeling of an unbounded number of sessions of security protocols may allow false attacks. On the other hand, we have proven our modeling of a bounded number of sessions to be precise.

- Our modelling is fully abstract for arbitrary cryptographic primitives that can be modeled as a convergent rewrite system that has the *finite variant property* [Comon-Lundh and Delaune 2005; Escobar et al. 2012]. This allows us to handle a larger class of cryptographic primitives than in Tiu and Dawson [2010], Cheval et al. [2010], Cheval et al. [2011], Baudet [2005], and Chevalier and Rusinowitch [2010]. Following our work, the recent work by Santiago et al. [2014] also provides support for rewrite systems that have the *finite variant property*. They additionally cover associative-commutative theories, even though their experimental evaluation suggests that these theories yield frequent non-termination problems for the associative-commutative theories. Moreover, they only provide support for a restricted class of processes. We were also able to show termination of our procedure for the sub-class of subterm convergent rewrite systems. Please note that deducability and hence static equivalence is undecidable, even for the class of optimally reducing convergent rewrite systems [Anantharaman et al. 2007]. Optimally reducing convergent rewrite theories generalize subterm convergent rewrite systems, while maintaining the finite variant property. Moreover, even though our termination proof does not apply, our tool terminated on specific protocols whose cryptographic primitives can be modeled as a convergent rewrite theories. These included the electronic voting protocols by Okamoto [1997] and Fujioka et al. [1992], which use trapdoor commitment and blind signature, respectively.
- Our procedure is implemented in the Active Knowledge in Security protocols (AKiSs) prototype tool and we used it among others to successfully prove anonymity in an electronic voting protocol [Fujioka et al. 1992]. For this electronic voting protocol, this is the first automated proof.

An extended abstract of the article [Chadha et al. 2012] authored by R. Chadha, S. Kremer, and Ș. Ciobâcă appeared in the European Symposium of Programming in 2012. In addition to the proofs that were not present in the extended abstract, this article also contains the proof of termination for subterm convergent rewrite theories. The proof of termination is by V. Cheval.

2. PRELIMINARIES

We recall some standard definitions and establish some notations that we shall be using throughout the article.

2.1. Terms

Let $\mathcal{F}$ be a signature, that is, a finite set of function symbols, and let ar be a function that assigns to each function symbol a natural number. Given a function symbol $f \in \mathcal{F}$, we say $ar(f) \in \mathbb{N}$ is the arity of f . 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 $\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n$, we denote the set $\mathcal{T}_{\mathcal{F}, \cup_{1 \leq i \leq n} \mathcal{A}_i}$ by $\mathcal{T}_{\mathcal{F}, \mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n}$. We assume that we have the following countably infinite pairwise disjoint sets: a set $\mathcal{N}$ of *private names*, a set $\mathcal{M}$ 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 channel (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 $w_1, w_2, \dots$ of the elements of $\mathcal{W}$. We let x, y, z range over $\mathcal{X}$. We also define the following sets.

Definition 2.1. The set $\mathcal{T}_{\mathcal{F}, \mathcal{N}, \mathcal{M}, \mathcal{W}, \mathcal{X}}$, denoted **Terms**, is the set of all terms, the set $\mathcal{T}_{\mathcal{F}, \mathcal{N}, \mathcal{M}}$, denoted **Messages**, is the set of *messages*, and the set $\mathcal{T}_{\mathcal{F}, \mathcal{N}, \mathcal{M}, \mathcal{X}}$, denoted **SMessages**, is the set of *symbolic messages*.

If t is a term, then we denote by $\text{vars}(t)$ the set of variables appearing in t , by $\text{names}(t)$ the set of names (public or private) appearing in t , and by $\text{st}(t)$ the set of all subterms of t . The functions vars , names , and st are extended as expected to sequences and sets of terms. A *position* is a string of positive natural numbers, and ϵ denotes the empty string. The set $\text{pos}(t)$ of positions of a term t is defined as usual [Baader and Nipkow 1998]. If $p \in \text{pos}(t)$, then $t|_p$ is the subterm of t at position p .

Example 2.2. Consider the signature $\mathcal{F} = \{\text{enc}, \text{dec}, \text{pair}, \text{fst}, \text{snd}\}$ where $\text{ar}(\text{enc}) = 3$, $\text{ar}(\text{dec}) = \text{ar}(\text{pair}) = 2$ and $\text{ar}(\text{fst}) = \text{ar}(\text{snd}) = 1$. The term $t = \text{pair}(\text{enc}(a, k_1, r_1), \text{enc}(b, k_2, r_2))$ models the pair of the encryptions of public names a and b with keys k_1 , respectively, k_2 , and randomness r_1 , respectively, r_2 . The set of positions $\text{pos}(t) = \{\epsilon, 1, 11, 12, 13, 2, 21, 22, 23\}$ and $t|_\epsilon = t$, $t|_1 = \text{enc}(a, k_1, r_1)$ and $t|_{23} = r_2$.

Substitutions. A substitution is a *partial* function $\sigma : \mathcal{W} \cup \mathcal{X} \rightarrow \text{Terms}$. We only consider substitutions that map elements of $\mathcal{W}$ to elements in **Messages** and elements of $\mathcal{X}$ to elements of **SMessages**. The domain of σ , denoted by $\text{dom}(\sigma)$, is defined as usual: $\text{dom}(\sigma) = \{o \in \mathcal{W} \cup \mathcal{X} \mid \sigma(o) \neq o\}$. For our purposes, we only consider substitutions with finite domains. We let $\text{range}(\sigma) = \{\sigma(u) \in \mathcal{T} \mid u \in \text{dom}(\sigma)\}$. If $\text{dom}(\sigma) = \{u_1, u_2, \dots, u_n\}$ and $t_i = \sigma(u_i)$ for each $1 \leq i \leq n$, then we shall write σ as $\{u_1 \mapsto t_1, \dots, u_n \mapsto t_n\}$. σ is said to be *ground* if $\text{range}(\sigma) \subseteq \text{Messages}$. The notation $\text{names}(\sigma)$ will denote the set $\text{names}(\text{range}(\sigma))$. A substitution σ can be extended to a (total) function $\sigma_{\text{ext}} : \mathcal{W} \cup \mathcal{X} \rightarrow \text{Terms}$ by letting $\sigma_{\text{ext}}(x) = x$ if $x \notin \text{dom}(\sigma)$ and $\sigma_{\text{ext}}(x) = \sigma(x)$ if $x \in \text{dom}(\sigma)$. As usual, σ extends homomorphically to a function $\text{apply}_\sigma : \text{Terms} \rightarrow \text{Terms}$ obtained by “applying” σ_{ext} homomorphically. Given $t \in \text{Terms}$, we denote $\text{apply}_\sigma(t)$ by $t\sigma$. If σ is a substitution and $X \subseteq \mathcal{W} \cup \mathcal{X}$, then we denoted by $\sigma[X]$ the substitution whose domain is restricted at most to X . Given two substitutions σ and τ , the substitution obtained by *composing* σ and τ , denoted $\sigma\tau$, is the unique substitution such that $\sigma\tau(x) = (\sigma(x))\tau$ for all $x \in \mathcal{W} \cup \mathcal{X}$.

2.2. 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 mgu a function that associates to any two unifiable terms s and t a most general unifier σ of s and t such that $\sigma = \sigma[\text{vars}(s, t)]$. It is well known [Baader and Nipkow 1998; Baader and Snyder 2001] that for any two unifiable terms s and t , there is a most general unifier, unique up to variable renaming.

A rewrite system R is a set of rewrite rules of the form $\ell \rightarrow r$ where $\ell, r \in \text{Messages}$, $\text{names}(\ell, r) = \emptyset$ and $\text{vars}(r) \subseteq \text{vars}(\ell)$. A term t can be rewritten in one step to u , denoted $t \rightarrow_R u$, if there exist a position $p \in \text{pos}(t)$, a rule $\ell \rightarrow r$ in R and a substitution σ such that $t|_p = \ell\sigma$ and u is obtained from t by replacing the subterm $t|_p$ by $r\sigma$. $\rightarrow_R^*$ denotes the transitive and reflexive closure of $\rightarrow_R$. A rewrite system is said to be *confluent* if for any t, t_1, t_2 such that $t \rightarrow_R^* t_1$ and $t \rightarrow_R^* t_2$ there exists u such that $t_1 \rightarrow_R^* u$ and $t_2 \rightarrow_R^* u$. A rewrite system is said to be *terminating* if it does not admit any infinite

sequence $t_0 \rightarrow_R t_1 \rightarrow_R t_2 \rightarrow_R \dots$. It is said to be *convergent* if it is both confluent and terminating. In a convergent rewrite system R , for every term t there is a unique term t' such that $t \rightarrow_R^* t'$ and there is no u such that $t' \rightarrow_R u$. t' is said to be the *normal form* of t . We denote by $t \downarrow_R$ the normal form of the term t . Two terms s and t are said to be equal modulo R , written $s =_R t$, if $s \downarrow_R = t \downarrow_R$. Given a substitution σ , we denote by $\sigma \downarrow_R$ a substitution such that $\text{dom}(\sigma \downarrow_R) \subseteq \text{dom}(\sigma)$ and for all $u \in \text{dom}(\sigma)$, $\sigma \downarrow_R(u) = \sigma(u) \downarrow_R$.

Example 2.3. Continuing Example 2.2, consider the rewrite system $R = \{\text{dec}(\text{enc}(x, y, z), y) \rightarrow x, \text{fst}(\text{pair}(x, y)) \rightarrow x, \text{snd}(\text{pair}(x, y)) \rightarrow y\}$. The first rewrite rule models that a message can be decrypted, provided decryption uses the same key (represented by variable y) as encryption. The two last rules model projection of the first and second component of a pair. Then we have that $t = \text{fst}(\text{pair}(\text{dec}(\text{enc}(a, k, r), k), b)) \rightarrow_R \text{fst}(\text{pair}(a, b)) \rightarrow_R a = t \downarrow_R$.

We recall the notions of *optimally reducing* [Narendran et al. 1997] and *subterm convergent* [Abadi and Cortier 2006] rewrite systems.

Definition 2.4. A rewrite system R is said to be *optimally reducing* if for any $\ell \rightarrow r \in R$ and any substitution θ such that all proper subterms of $\ell\theta$ are in normal form, we have that $r\theta$ is in normal form.

Definition 2.5. A rewrite system R is said to be *subterm convergent* if R is convergent and for each rule $\ell \rightarrow r \in R$, we have that either $r \in \text{st}(\ell)$ or r is a constant.

We immediately note that any subterm convergent rewrite system R can be easily converted into an equivalent optimally reducing rewrite system by replacing every rewrite rule $\ell \rightarrow r$ in R by $\ell \rightarrow r \downarrow_R$.

Example 2.6. The rewrite system $R = \{\text{dec}(\text{enc}(x, y, z), y) \rightarrow x, \text{fst}(\text{pair}(x, y)) \rightarrow x, \text{snd}(\text{pair}(x, y)) \rightarrow y\}$ given in Example 2.3 is subterm convergent. We shall give examples of convergent rewrite systems that are not subterm convergent when we discuss the case studies on electronic voting in Section 6.2.

Remark 2.7. When R is clear from the context or unimportant we will simply drop the subscript R in $\rightarrow_R$ and $\downarrow_R$.

2.3. The Finite Variant Property

Given a convergent rewrite system, we now define the notion of complete set of variants, which was introduced by Comon-Lundh and Delaune [2005]. Our notion is slightly stronger than the notion defined in Comon-Lundh and Delaune [2005] and was first introduced in Escobar et al. [2012]. See Cholewa et al. [2014] for a comparison of the various definitions of variants.

Definition 2.8. A set of substitutions $\text{variants}(t_1, \dots, t_k)$ is called a *complete set of variants* of the terms $t_1, \dots, t_k$ if for any substitution ω there exist $\sigma \in \text{variants}(t_1, \dots, t_k)$ and a substitution τ such that for all $1 \leq j \leq k$ we have that $\omega[\text{vars}(t_j)] \downarrow = (\sigma \downarrow \tau)[\text{vars}(t_j)]$ and $(t_j \omega) \downarrow = (t_j \sigma) \downarrow \tau$.

Intuitively, if $\text{variants}(t) = \{\sigma_1, \sigma_2, \dots, \sigma_k\}$, then the set of terms $\text{preterms}(t) = \{t\sigma_1 \downarrow, t\sigma_2 \downarrow, \dots, t\sigma_k \downarrow\}$ represent pre-computations of all possible instances of t in the following sense: If ω is a substitution and t_ω is the term $t\omega \downarrow$, then there is a term $t' \in \text{preterms}(t)$ and a substitution τ such that $t'\tau$ is the *syntactic* term t_ω . No rewrite rules are needed to compute t_ω from $t'\tau$.

Example 2.9. Consider the rewrite system introduced in Example 2.3 and let $t = \text{dec}(\text{fst}(x), y)$. We have that $\text{variants}(t) = \{\emptyset, \sigma_1, \sigma_2\}$ where $\emptyset$ denotes the identity

substitution and

$$\begin{aligned}\sigma_1 &= \{x \mapsto \text{pair}(z_1, z_2)\}, \text{ and} \\ \sigma_2 &= \{x \mapsto \text{pair}(\text{enc}(z_1, y, z_2), z_3)\}.\end{aligned}$$

Intuitively, the substitution $\emptyset$ covers the cases where both the decryption and projection may fail; σ_1 corresponds to the situation where the projection succeeds, but decryption may fail; and σ_2 accounts for the situations where both projection and decryption succeed. Note that the case where projection fails and decryption succeeds is not possible.

In Ciobăcă [2011], Ciobăcă presents an algorithm for computing such complete sets of variants that is correct whenever the rewrite system is *optimally reducing* [Narendran et al. 1997]. Optimally reducing rewrite systems include subterm convergent systems [Abadi and Cortier 2006] (and hence the classical Dolev Yao theories for encryption, signatures, and hash functions), as well as a theory for modeling blind signatures [Kremer and Ryan 2005]. Moreover, complete sets of variants can be used to perform unification modulo R [Escobar et al. 2012; Ciobăcă 2011].

Definition 2.10. Given sets of terms $\{s_i\}_{i \in I}$ and $\{t_i\}_{i \in I}$, let $X = \text{vars}(\{s_i, t_i\}_{i \in I})$. A set of substitutions $\text{mgu}_R(\{s_i \stackrel{?}{=} t_i\}_{i \in I})$ is called a *complete set of unifiers modulo R* of the system of equations $\{s_i \stackrel{?}{=} t_i\}_{i \in I}$ if each of the following holds:

- (1) $\text{dom}(\sigma) \subseteq \text{vars}(X)$ for each $\sigma \in \text{mgu}_R(\{s_i \stackrel{?}{=} t_i\}_{i \in I})$
- (2) $s_i \sigma =_R t_i \sigma$ for each $i \in I$ and for each $\sigma \in \text{mgu}_R(\{s_i \stackrel{?}{=} t_i\}_{i \in I})$.
- (3) For any substitution θ such that $s_i \theta =_R t_i \theta$ for every $i \in I$, there exists a substitution $\sigma \in \text{mgu}_R(\{s_i \stackrel{?}{=} t_i\}_{i \in I})$ and a substitution τ with $\theta[X] =_R (\sigma \tau)[X]$.

For singleton systems, we also write $\text{mgu}_R(s, t)$ instead of $\text{mgu}_R(\{s \stackrel{?}{=} t\})$.

For the remainder of the article, we assume that the rewrite system is convergent and has the finite variant property.

2.4. Frames, Deducibility, and Static Equivalence

Recall that we have fixed an enumeration $w_1, w_2, \dots$ of the elements of the set $\mathcal{W}$. As in Abadi and Fournet [2001], we will use the notion of a *frame* to represent messages that have been recorded by the attacker.

Definition 2.11. A *frame* φ is a substitution $\{w_1 \mapsto t_1, \dots, w_n \mapsto t_n\}$ where $t_i \in \text{Messages}$ ($1 \leq i \leq n$).

Intuitively, w_i in a frame points to the i th message recorded by the attacker in a protocol run. Note that in our definition, every frame with $|\text{dom}(\varphi)| = n$ has $\text{dom}(\varphi) = \{w_1, \dots, w_n\}$. We denote the set of all frames as Frames . The adversary can use the public names as well as recorded messages to construct new messages. This is modeled as the deducibility relation.

Definition 2.12. 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 $t \in \text{Messages}$ and $r\varphi =_R t$. We write Recipes for the set $\mathcal{T}_{\mathcal{F}, \mathcal{M}, \mathcal{W}}$.

Intuitively, the recipe r tells how the attacker can construct the message t from the recorded messages. Note that the same term t can be constructed using different recipes. A frame $\varphi' = \{w_1 \mapsto t'_1, \dots, w_m \mapsto t'_m\}$ *extends* a frame $\varphi = \{w_1 \mapsto t_1, \dots, w_n \mapsto t_n\}$ if $m \geq n$ and if $t'_i = t_i$ for all $1 \leq i \leq n$. It is easy to see that if φ' extends φ and if $\varphi \vdash^r t$, then $\varphi' \vdash^r t$.

Example 2.13. Consider the signature $\mathcal{F}$ and the rewrite system R in Example 2.3. Let $\varphi = \{w_1 \mapsto \text{enc}(s, k, r), w_2 \mapsto k\}$ where $s, k, r \in \mathcal{N}$ are private names. Then we have that $\varphi \vdash^{\text{dec}(w_1, w_2)} s$. Note that $\text{dec}(w_1, k) \notin \text{Recipes}$ as $k \in \mathcal{N}$. If we had that $s \in \mathcal{M}$, then we would also have that $\varphi \vdash^s s$ reflecting that public names are always deducible.

We now recall *static equivalence* of frames [Abadi and Fournet 2001] to capture indistinguishability of frames. Recall that two terms can be indistinguishable to an attacker even if the two terms are distinct. For example, 0 encrypted using a symmetric key unknown to the attacker and 1 encrypted using the same key are indistinguishable to the attacker. Thus, instead of checking of direct equality between messages, the attacker can perform a series of tests to distinguish between two frames. This is the intuition behind the following definition:

Definition 2.14. Let $r_1, r_2 \in \text{Recipes}$. 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 , that is, r_1 and r_2 are recipes for the same term in φ .

Frames φ_1 and φ_2 are *statically equivalent* (written $\varphi_1 \approx_s \varphi_2$) iff for all $r_1, r_2 \in \text{Recipes}$ we have that $(r_1 = r_2)\varphi_1$ iff $(r_1 = r_2)\varphi_2$.

Example 2.15. Let $a, b \in \mathcal{M}$ and $r, k, k' \in \mathcal{N}$. We have that $\{w_1 \mapsto \text{enc}(a, k, r), w_2 \mapsto k\} \not\approx_s \{w_1 \mapsto \text{enc}(b, k, r), w_2 \mapsto k\}$ because the test $(\text{dec}(w_1, w_2) \stackrel{?}{=} a)$ distinguishes the two frames. However, $\{w_1 \mapsto \text{enc}(a, k, r), w_2 \mapsto k'\} \approx_s \{w_1 \mapsto \text{enc}(b, k, r), w_2 \mapsto k'\}$.

3. A CRYPTOGRAPHIC PROCESS CALCULUS

We shall assume that cryptographic protocols are modeled using a simple process calculus that has similarities with the applied pi calculus [Abadi and Fournet 2001]. The applied pi calculus has proven to be useful for specifying and verifying cryptographic protocols; there are tools that automate verification of protocols in this model [Blanchet 2001]. We shall further restrict our attention to the finite, that is, replication-free fragment of applied pi calculus. This restriction is important because observational equivalence becomes undecidable with replication [Hüttel 2002]. With this restriction, one can model a bounded number of protocol instances.

In this section we define our process calculus. We begin by defining its syntax.

Syntax. Recall that we have fixed a first-order signature $\mathcal{F}$, 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 (see Section 2). The terms of the set $\mathcal{T}_{\mathcal{F}, \mathcal{N}, \mathcal{M}, \mathcal{W}, \mathcal{X}}$ are also identified modulo a fixed subterm convergent rewrite system R (see Section 2).

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 (note that here **in** and **out** are fresh symbols not occurring in $\mathcal{F}$):

$$\begin{array}{ll} a ::= \mathbf{in}(c, x) & \text{receive action} \\ & \mathbf{out}(c, t) \quad \text{send action} \\ & [s \stackrel{?}{=} t] \quad \text{test action,} \end{array}$$

where $x \in \mathcal{X}$, $s, t \in \text{SMessages}$, $c \in \mathcal{C}$. A *trace* T is a sequence of actions $T = a_1.a_2.\dots.a_n$. As usual, a receive action $\mathbf{in}(c, x)$ acts as a binding construct for the variable 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 GndTraces . We also assume the usual definition of a name occurring in a trace.

A process P is defined to be a set of traces $P = \{T_1, \dots, T_n\}$. We say that a process is ground if all of its traces are ground. We identify traces with singleton processes.

Remark 3.1. Contrary to the applied pi calculus [Abadi and Fournet 2001] we do not have an ν operator for generating new names: As we only consider a finite number of sessions, we can simply use 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, we note that 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 Recipes . Recall that the set Recipes is the set $\mathcal{T}_{\mathcal{F}, \mathcal{M}, \mathcal{W}}$ (see Section 2). The set of labels, Labels , is

$$\text{Labels} = \{\mathbf{in}(c, r), \mathbf{out}(c), \mathbf{test} \mid r \in \text{Recipes}, c \in \mathcal{C}\}.$$

The labeled transition system T is a subset of $(\text{GndTraces} \times \text{Frames}) \times \text{Labels} \times (\text{GndTraces} \times \text{Frames})$, and we shall write $(T, \varphi) \xrightarrow{\ell} (T', \varphi')$ whenever $((T, \varphi), \ell, (T', \varphi')) \in T$. 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:

$$\begin{aligned} \text{RECEIVE} & \frac{\varphi \vdash^r t}{(\mathbf{in}(c, x).T, \varphi) \xrightarrow{\mathbf{in}(c, r)} (T\{x \mapsto t\}, \varphi)} \\ \text{SEND} & \frac{}{(\mathbf{out}(c, t).T, \varphi) \xrightarrow{\mathbf{out}(c)} (T, \varphi \cup \{\mathbf{w}_{|\text{dom}(\varphi)|+1} \mapsto t\})} \\ \text{TEST} & \frac{s = \mathbf{R}t}{([s=t].T, \varphi) \xrightarrow{\mathbf{test}} (T, \varphi)}. \end{aligned}$$

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 the transition rule SEND records the message sent in the frame. Finally, the rule TEST is an internal action.

As usual, we shall write $(T_0, \varphi_0) \xrightarrow{\ell_1, \dots, \ell_n} (T_n, \varphi_n)$ when $(T_0, \varphi_0) \xrightarrow{\ell_1} (T_1, \varphi_1) \dots \xrightarrow{\ell_n} (T_n, \varphi_n)$, and we say that $\ell_1 \dots \ell_n$ is a *run* of (T_0, φ_0) . We shall write $(T, \varphi) \xRightarrow{\ell} (T', \varphi')$ 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 $\mathbf{test}$ actions. We write $(T, \varphi) \xRightarrow{\ell_1, \dots, \ell_n} (T_n, \varphi_n)$ when $(T, \varphi) \xrightarrow{\ell_1} (T_1, \varphi_1) \xrightarrow{\ell_2} \dots \xrightarrow{\ell_n} (T_n, \varphi_n)$. If $P = \{T_1, \dots, T_m\}$ is a process, then we write $(P, \varphi) \xrightarrow{\ell_1, \dots, \ell_n} (T', \varphi')$ (respectively, $\xRightarrow{\ell_1, \dots, \ell_n} (T', \varphi')$) if there exists a trace $T \in P$ such that $(T, \varphi) \xrightarrow{\ell_1, \dots, \ell_n} (T', \varphi')$ (respectively, $(T, \varphi) \xRightarrow{\ell_1, \dots, \ell_n} (T', \varphi')$).

Process equivalences. In this section we will define various flavors of trace equivalence that will be useful in this article. We first recall the standard definition of trace equivalence in cryptographic process algebras.

Definition 3.2 (Trace Equivalence). A ground process P is said to be *trace-included* in a ground process Q (written $P \sqsubseteq_t Q$) if, whenever $(P, \emptyset) \xRightarrow{\ell_1, \dots, \ell_n} (T, \varphi)$, there exist T', φ'

such that $(Q, \emptyset) \xrightarrow{\ell_1, \dots, \ell_n} (T', \varphi')$ 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. The coarser trace equivalence, which we denote by $\approx_{ct}$, is the trace equivalence that can actually be verified by our procedure.

Definition 3.3 (Coarse Trace Equivalence). Given ground processes P and Q , we say that $P \sqsubseteq_{ct} Q$ if, whenever $(P, \emptyset) \xrightarrow{\ell_1, \dots, \ell_n} (T, \varphi)$ and $(r_1 = r_2)\varphi$ for some recipes r_1, r_2 , there exist T', φ' such that $(Q, \emptyset) \xrightarrow{\ell_1, \dots, \ell_n} (T', \varphi')$ and $(r_1 = r_2)\varphi'$. 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 3.4. Let P and Q be the ground processes defined as follows:

$$\begin{aligned} P &= \{ \mathbf{out}(c, a). \mathbf{out}(c, a) \} \text{ and} \\ Q &= \{ \mathbf{out}(c, a). \mathbf{out}(c, a), \mathbf{out}(c, a). \mathbf{out}(c, b) \}. \end{aligned}$$

Clearly, $P \sqsubseteq_{ct} Q$. Observe also that $Q \sqsubseteq_{ct} P$. Indeed, only trivial equalities hold on the frame $\{w_1 \mapsto a, w_2 \mapsto b\}$, and therefore these also hold on $\{w_1 \mapsto a, w_2 \mapsto a\}$. Thus, we have that $P \approx_{ct} Q$ while $P \not\approx_t Q$.

We will, however, show that these two notions coincide for a class of *determinate processes*. In the context of the applied pi calculus, determinate processes were previously studied by Cortier and Delaune [2009].

Definition 3.5 (Determinate Process). We say that a ground process P is *determinate* if, whenever $(P, \emptyset) \xrightarrow{\ell_1, \dots, \ell_n} (T, \varphi)$ and $(P, \emptyset) \xrightarrow{\ell_1, \dots, \ell_n} (T', \varphi')$, 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. The following is immediate from the definition.

PROPOSITION 3.6. *A ground trace, that is, a ground process consisting of single trace, is determinate.*

As already mentioned above, it was demonstrated in Cortier and Delaune [2009] 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 3.7. *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 $P \approx_t Q$.*

PROOF.

($\Rightarrow$) Follows immediately from definition of $\approx_t$ and $\approx_{ct}$.

($\Leftarrow$) Let P and Q be determinate processes. We need to show that $P \approx_{ct} Q$ implies $P \approx_t Q$. We proceed by contradiction. Suppose that $P \approx_{ct} Q$ and $P \not\approx_t Q$. We suppose $P \not\sqsubseteq_t Q$ (the case of $Q \not\sqsubseteq_t P$ being symmetric). As $P \not\sqsubseteq_t Q$, we have that

there exist $\ell_1, \dots, \ell_n, T, \varphi$, such that $(P, \emptyset) \xrightarrow{\ell_1, \dots, \ell_n} (T, \varphi)$ and

(1) either there exist no φ', T' such that $(Q, \emptyset) \xrightarrow{\ell_1, \dots, \ell_n} (T', \varphi')$,

(2) or for all φ', T' such that $(Q, \emptyset) \xrightarrow{\ell_1, \dots, \ell_n} (T', \varphi')$ we have that $\varphi \not\approx_s \varphi'$.

In the first case, we have that $P \not\approx_{ct} Q$, contradicting our hypothesis. In the second case, as $\varphi \not\approx_s \varphi'$, there exist r, r' such that $(r = r')\varphi$ and $(r \neq r')\varphi'$ (or, vice versa, the other case is symmetric). As $P \sqsubseteq_{ct} Q$, we have that there exist T'', φ'' such that

$(Q, \emptyset) \xrightarrow{\ell_1, \dots, \ell_n} (T'', \varphi'')$ and $(r = r')\varphi''$. As Q is determinate, we have that $\varphi' \approx_s \varphi''$. This yields a contradiction, as $(r \neq r')\varphi'$ and $(r = r')\varphi''$ would imply $\varphi' \not\approx_s \varphi''$. $\square$

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

Definition 3.8 (Fine-Grained Trace Equivalence). Given ground processes P and Q , we say that $P \sqsubseteq_{ft} Q$ if for each 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 from the definition that $\approx_{ft} \subseteq \approx_t$. The difference between these two relations is illustrated by the following example.

Example 3.9. Let P and Q be ground processes defined as follows:

$$P = \{ \text{out}(c, \text{enc}(a, k)).\text{out}(c, \text{enc}(b, k)).\text{in}(c, x).[x = \text{enc}(a, k)].\text{out}(c, k), \\ \text{out}(c, \text{enc}(a, k)).\text{out}(c, \text{enc}(b, k)).\text{in}(c, x).[x = \text{enc}(b, k)].\text{out}(c, k) \}$$

$$Q = \{ \text{out}(c, \text{enc}(a, k)).\text{out}(c, \text{enc}(b, k)).\text{in}(c, x).[x = \text{enc}(\text{dec}(x, k), k)].\text{out}(c, k) \},$$

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

As already mentioned, our procedure is able to check $\approx_{ct}$, which coincides with $\approx_t$ when processes are determinate. In the case where processes are not determinate we can use our procedure to check $\approx_{ct}$ and $\approx_{ft}$ in order to over- and under-approximate $\approx_t$. Indeed, as traces are determinate processes a procedure for checking $\approx_{ct}$ can be used to verify $\approx_{ft}$.

4. MODELING TRACES AS HORN CLAUSES

Our decision procedure is based on a fully abstract modelling of a trace in first-order Horn clauses. We give the details of this modelling; 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

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

and the set of *symbolic runs*, SRuns , as the set of finite sequences of symbolic labels (see Figure 1). The empty sequence is denoted by ϵ . Sometimes we simply 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 SRecipes . We use capital letters X, Y, Z to range over $\mathcal{Y}$. Intuitively, a symbolic recipe stands for a set of recipes.

We extend the definition of substitutions to include variables from $\mathcal{Y}$ in its domain. However, we only consider substitutions that map variables in $\mathcal{Y}$ to SRecipes . A ground substitution must map variables in $\mathcal{Y}$ to Recipes . The notion of most general unifiers is extended to symbolic recipes as expected.

Predicates. The predicates used in our modelling and the semantics of the predicates are given in Figure 1. The ground predicates are interpreted over a pair: a trace T and a frame φ . A predicate P with free variables is interpreted over a triple: a trace T , a frame φ , and a substitution σ :

$$(T, \varphi_0, \sigma) \models P \text{ iff } (T, \varphi_0) \models P\sigma.$$

Fig. 1. Predicates.

We consider 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, that is, does not block due to a test that fails. The *intruder knowledge predicate* $k_w(R, t)$ 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* $i_w(R, R')$ says that whenever the (symbolic) run w is executed, the (symbolic) recipes R and R' are recipes for the same (symbolic) term. Observe that the term t in the definition of the predicate $i_w(R, R')$, if it exists, must be unique (modulo R). The *reachable identity predicate* $ri_w(R, R')$ is a short form for the conjunction of the predicates r_w and $i_w(R, R')$.

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.

Note that in case f is a ground formula, we shall omit σ as we do not need the substitution σ . If, in addition to f being ground, we have that $\text{dom}(\varphi) = \emptyset$, then 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 will take the form of Horn clauses, and we shall be mainly concerned with them.

Definition 4.1. A *statement* is a Horn clause of the form $H \Leftarrow B_1, \dots, B_n$ where:

- (1) $H \in \{r_{\ell_1, \dots, \ell_k}, k_{\ell_1, \dots, \ell_k}(R, t), i_{\ell_1, \dots, \ell_k}(R, R'), ri_{\ell_1, \dots, \ell_k}(R, R')\}$.
- (2) For each $1 \leq i \leq n$, $B_i = k_{\ell_1, \dots, \ell_{j_i}}(X_i, t_i)$

for some $l_1, \dots, l_k \in \text{SLabels}$, $t \in \text{SMessages}$, $R, R' \in \text{SRecipes}$, $j_i \leq k$, $t_1, \dots, t_n \in \text{SMessages}$, and $X_1, \dots, X_n \in \mathcal{Y}$. Furthermore $X_1, \dots, X_n$ are distinct variables and if $H = k_{\ell_1, \dots, \ell_k}(R, t)$ then $\text{vars}(t) \subseteq \text{vars}(t_1, \dots, t_n)$.

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

Remark 4.2. We sometimes abuse language and call σ a closing substitution for a statement $H \Leftarrow B_1, \dots, B_n$ if σ is closing for each of the formulas $H, B_1, \dots, B_n$.

Remark 4.3. Let $f = H \Leftarrow B_1, \dots, B_n$ be a statement.

- f is said to be a *reachability statement* if H is of the form $r_{l_1, \dots, l_k}$.
- f is said to be a *deduction statement* if H is of the form $k_{l_1, \dots, l_k}(R, t)$.
- f is said to be an *equational statement* if H is of the form $l_{l_1, \dots, l_k}(R, R')$.
- f is said to be a *reachable identity statement* if H is of the form $rl_{l_1, \dots, l_k}(R, R')$.

4.1. The Set of Seed Statements

As mentioned above, our decision procedure is based on a fully abstract modelling of a trace in first-order Horn clauses. In this section, given a trace T , we will give a set of statements $\text{seed}(T)$ that will serve as a starting point for the modelling. We shall also establish that the set of statements $\text{seed}(T)$ is a sound and (partially) complete abstraction of the trace T . In order to formally define $\text{seed}(T)$, we start by fixing some notational conventions.

Let $T = a_1.a_2 \dots a_n$ be a ground trace. We assume w.l.o.g. the following naming conventions:

- (1) If a_i is a receive action then $a_i = \mathbf{in}(c_i, x_i)$.
- (2) $x_i \neq x_j$ for any $i \neq j$.
- (3) If a_i is a send action then $a_i = \mathbf{out}(c_i, t_i)$.
- (4) If a_i is a test action then $a_i = [s_i \stackrel{?}{=} t_i]$.

Moreover, for each $1 \leq i \leq n$, let $\ell_i \in \text{SLabels}$ 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 $0 \leq m \leq n$, let the sets $\text{Rcv}_T(m)$, Send_T and $\text{Test}_T(m)$, respectively, denote the indices of the receive actions, send actions, and test actions among $a_1, \dots, a_m$. Formally,

$$\begin{cases} \text{Rcv}_T(m) = \{i \mid 1 \leq i \leq m, a_i = \mathbf{in}(c_i, x_i)\} \\ \text{Send}_T(m) = \{i \mid 1 \leq i \leq m, a_i = \mathbf{out}(c_i, t_i)\} \\ \text{Test}_T(m) = \{i \mid 1 \leq i \leq m, a_i = [s_i \stackrel{?}{=} t_i]\} \end{cases}$$

Given a set of public names $\mathcal{M}_0 \subseteq \mathcal{M}$, the *set of seed statements* associated to T and $\mathcal{M}_0$, denoted $\text{seed}(T, \mathcal{M}_0)$, is defined to be the set of statements given in Figure 2. If $\mathcal{M}_0 = \mathcal{M}$, then $\text{seed}(T, \mathcal{M})$ is said to be the set of seed statements associated to T and in this case we write $\text{seed}(T)$ as a shortcut for $\text{seed}(T, \mathcal{M})$.

Remark 4.4. Please note that while constructing the set of seed statements, we apply the most general unifier modulo $\mathbf{R}$ to all tests. In addition, we also apply finite variants. This allows us to *get rid* of rewriting in our procedure.

Example 4.5. As an example consider the signature $\mathcal{F} = \{\text{pair}, \text{fst}, \text{snd}, \text{h}, \text{a}\}$ where $\text{ar}(\text{pair}) = \text{ar}(\text{h}) = 2$, $\text{ar}(\text{fst}) = \text{ar}(\text{snd}) = 1$, and $\text{ar}(\text{a}) = 0$ equipped with the rewrite

$$\begin{aligned}
& r_{\ell_1 \sigma \tau \downarrow, \dots, \ell_m \sigma \tau \downarrow} \Leftarrow \{k_{\ell_1 \sigma \tau \downarrow, \dots, \ell_{j-1} \sigma \tau \downarrow}(X_j, x_j \sigma \tau \downarrow)\}_{j \in \text{Rcv}_T(m)} \\
& \quad \text{for all } 0 \leq m \leq n \\
& \quad \text{for all } \sigma \in \text{mgu}_R(\{s_k = t_k\}_{k \in \text{Test}_T(m)}) \\
& \quad \text{for all } \tau \in \text{variants}(\ell_1 \sigma, \dots, \ell_m \sigma) \\
& k_{\ell_1 \sigma \tau \downarrow, \dots, \ell_m \sigma \tau \downarrow}(\text{w}|\text{Send}_T(m)|, t_m \sigma \tau \downarrow) \Leftarrow \{k_{\ell_1 \sigma \tau \downarrow, \dots, \ell_{j-1} \sigma \tau \downarrow}(X_j, x_j \sigma \tau \downarrow)\}_{j \in \text{Rcv}_T(m)} \\
& \quad \text{for all } m \in \text{Send}_T(n) \\
& \quad \text{for all } \sigma \in \text{mgu}_R(\{s_k = t_k\}_{k \in \text{Test}_T(m)}) \\
& \quad \text{for all } \tau \in \text{variants}(\ell_1 \sigma, \dots, \ell_m \sigma, t_m \sigma) \\
& k(c, c) \Leftarrow \\
& \quad \text{for all public names } c \in \mathcal{M}_0 \\
& k_{\ell_1, \dots, \ell_m}(f(Y_1, \dots, Y_k), f(y_1, \dots, y_k) \tau \downarrow) \Leftarrow \{k_{\ell_1, \dots, \ell_m}(Y_j, y_j \tau \downarrow)\}_{j \in \{1, \dots, k\}} \\
& \quad \text{for all } 0 \leq m \leq n \\
& \quad \text{for all function symbols } f \text{ of arity } k \\
& \quad \text{for all } \tau \in \text{variants}(f(y_1, \dots, y_k)).
\end{aligned}$$

Fig. 2. Seed statements.

system $R = \{\text{fst}(\text{pair}(x, y)) \rightarrow x, \text{snd}(\text{pair}(x, y)) \rightarrow y\}$ and the trace

$$T = \mathbf{in}(c, x).[\text{fst}(x) \stackrel{?}{=} a].\mathbf{out}(c, h(s, \text{snd}(x))).\mathbf{out}(c, s).$$

Note that s is a private name. The set $\text{seed}(T, \emptyset)$ (ignoring public names) consists of the following clauses:

$$r_{\mathbf{in}(c, x)} \Leftarrow k(X, x), \quad (1)$$

$$r_{\mathbf{in}(c, \text{pair}(a, x)).\text{test}} \Leftarrow k(X, \text{pair}(a, x)), \quad (2)$$

$$r_{\mathbf{in}(c, \text{pair}(a, x)).\text{test.out}(c)} \Leftarrow k(X, \text{pair}(a, x)), \quad (3)$$

$$r_{\mathbf{in}(c, \text{pair}(a, x)).\text{test.out}(c).\mathbf{out}(c)} \Leftarrow k(X, \text{pair}(a, x)), \quad (4)$$

$$k_{\mathbf{in}(c, \text{pair}(a, x)).\text{test.out}(c)}(w_1, h(s, x)) \Leftarrow k(X, \text{pair}(a, x)), \quad (5)$$

$$k_{\mathbf{in}(c, \text{pair}(a, x)).\text{test.out}(c).\mathbf{out}(c)}(w_2, s) \Leftarrow k(X, \text{pair}(a, x)), \quad (6)$$

$$k_w(a, a) \Leftarrow, \quad (7)$$

$$k_w(\text{fst}(Y), \text{fst}(y)) \Leftarrow k_w(Y, y), \quad (8)$$

$$k_w(\text{fst}(Y), y_1) \Leftarrow k_w(Y, \text{pair}(y_1, y_2)), \quad (9)$$

$$k_w(\text{snd}(Y), \text{snd}(y)) \Leftarrow k_w(Y, y), \quad (10)$$

$$k_w(\text{snd}(Y), y_2) \Leftarrow k_w(Y, \text{pair}(y_1, y_2)), \quad (11)$$

$$k_w(\text{pair}(Y_1, Y_2), \text{pair}(y_1, y_2)) \Leftarrow k_w(Y_1, y_1), k_w(Y_2, y_2), \quad (12)$$

$$k_w(h(Y_1, Y_2), h(y_1, y_2)) \Leftarrow k_w(Y_1, y_1), k_w(Y_2, y_2), \quad (13)$$

where $w \in \{u \mid \exists v. uv = \mathbf{in}(c, \text{pair}(a, x)).\text{test.out}(c).\mathbf{out}(c)\}$.

We may note that in the first block of four reachability statements ((1)–(4)), in order to satisfy the test $[\text{fst}(x) \stackrel{?}{=} a]$, the attacker needs to be able to construct a pair $\text{pair}(a, x)$. This condition is obtained by computing $\text{mgu}_R(\{\text{fst}(x) = a\}) = \{x \mapsto \text{pair}(a, x)\}$. The second block of clauses adds a knowledge clause for each send action in the trace. The third block of clauses represents the attacker capabilities. It computes the set

of variants on $f(y_1, \dots, y_k)$ for each function symbol f in the signature, for example, $\text{variants}(\text{fst}(x)) = \{\emptyset, \{x \mapsto \text{pair}(x, y)\}\}$ (where $\emptyset$ denotes the identity substitution).

We shortly show that the set of seed statements is a sound and (partially) complete modelling of a trace. However, we need one more definition to state this fact.

Definition 4.6. Let K be a set of statements. We define $\mathcal{H}(K)$ to be the smallest set of ground terms such that:

$$\begin{aligned} f &= (H \Leftarrow B_1, \dots, B_n) \in K \\ \text{SIMPLE CONSEQUENCE} &\frac{\sigma \text{ grounding for } f \quad B_1\sigma \in \mathcal{H}(K) \dots B_n\sigma \in \mathcal{H}(K)}{H\sigma \in \mathcal{H}(K)} \\ \text{EXTENDK} &\frac{k_u(R, t) \in \mathcal{H}(K)}{k_{uv}(R, t) \in \mathcal{H}(K)}. \end{aligned}$$

(Equivalently, $\mathcal{H}(K)$ is the least Herbrand model of $K \cup \{k_{\ell_1, \dots, \ell_{n+1}}(X, x) \Leftarrow k_{\ell_1, \dots, \ell_n}(X, x)\}_{n \in \mathbb{N}}.$)

We show that as far as reachability predicates and intruder knowledge predicates are concerned, the set $\text{seed}(T)$ is a complete abstraction of a trace (please see the Electronic Appendix for the proof):

THEOREM 4.7. *Let T be a ground trace.*

—(Soundness.) *For any statement $f \in \text{seed}(T) \cup \mathcal{H}(\text{seed}(T))$, $T \models f$.*

—(Completeness.) *If $(T, \emptyset) \xrightarrow{L_1, \dots, L_m} (S, \varphi)$, then:*

(1) $\text{r}_{L_1\varphi\downarrow, \dots, L_m\varphi\downarrow} \in \mathcal{H}(\text{seed}(T)).$

(2) *If $\varphi \vdash^R t$ then $k_{L_1\varphi\downarrow, \dots, L_m\varphi\downarrow}(R, t\downarrow) \in \mathcal{H}(\text{seed}(T)).$*

Remark 4.8. Please note that the set $\text{seed}(T)$ is only partially complete in that we have not shown in Theorem 4.7 that if $\varphi \vdash^R t$ and $\varphi \vdash^{R'} t$, then $k_{L_1\varphi\downarrow, \dots, L_m\varphi\downarrow}(R, R') \in \mathcal{H}(\text{seed}(T)).$

We will shortly show how the completeness of $\text{seed}(T)$ can be built on to achieve (a) full abstraction of the trace T and (b) a procedure for checking equivalences $\approx_{ct}$ and $\approx_{ft}$.

5. PROCEDURE FOR DECIDING TRACE EQUIVALENCE

We shall now describe a procedure for deciding trace equivalence. At a high level, this consists of two steps.

- (1) A saturation procedure that constructs a set of *simple* statements from the set $\text{seed}(T)$, 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 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.

We shall now give the details of the procedure. We start by the saturation procedure.

5.1. Knowledge Bases and Saturation

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

Definition 5.1. Given a statement $f = H \Leftarrow B_1, \dots, B_n$,

$$\begin{array}{l}
\text{RESOLUTION} \frac{f \in K, g \in K_{\text{solved}}, \quad f = (H \Leftarrow k_{uv}(X, t), B_1, \dots, B_n) \\
g = (k_w(R, t') \Leftarrow B_{n+1}, \dots, B_m) \quad \sigma = \text{mgu}(k_u(X, t), k_w(R, t')) \quad t \notin \mathcal{X}}{K := K \oplus h \text{ where } h = ((H \Leftarrow B_1, \dots, B_m)\sigma)} \\
\\
\text{EQUATION} \frac{f, g \in K_{\text{solved}}, \quad f = (k_u(R, t) \Leftarrow B_1, \dots, B_n) \\
g = (k_{u'v'}(R', t') \Leftarrow B_{n+1}, \dots, B_m) \quad \sigma = \text{mgu}(k_u(-, t), k_{u'}(-, t'))}{K := K \oplus h \text{ where } h = ((i_{u'v'}(R, R') \Leftarrow B_1, \dots, B_m)\sigma)} \\
\\
\text{TEST} \frac{f = (i_u(R, R') \Leftarrow B_1, \dots, B_n) \quad f, g \in K_{\text{solved}}, \\
g = (r_{u'v'} \Leftarrow B_{n+1}, \dots, B_m) \quad \sigma = \text{mgu}(u, u')}{K := K \oplus h \text{ where } h = ((r_{i_{u'v'}}(R, R') \Leftarrow B_1, \dots, B_m)\sigma)}
\end{array}$$

Fig. 3. Saturation rules.

- f is said to be *solved* if for all $1 \leq i \leq n$, $B_i = k_{\ell_1, \dots, \ell_{j_i}}(X_i, x_i)$ for some variables $x_i \in \mathcal{X}$, $X_i \in \mathcal{Y}$.
- f is said to be *well-formed* if one of the following holds:

- (1) f is not solved.
- (2) f is a solved reachability, equational, and reachable identity statement.
- (3) f is a solved deduction statement such that if $H = k_{\ell_1, \dots, \ell_k}(R, t)$ then $t \notin \mathcal{X}$.

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

Given an initial knowledge base K , the saturation procedure produces another knowledge base $\text{sat}(K)$. The saturation procedure proceeds as follows. First new statements are *generated* and 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. Each of the rule generates a new statement h . The rule RESOLUTION applies the standard rule of resolution from first-order logic to an unsolved and a solved deduction statement and allows us to propagate constraints imposed from a partial execution of a trace to its possible extensions. The rule EQUATION allows us to derive new identities on recipes that may be imposed by the execution of the protocol. The rule TEST allows us to conclude which identities necessarily hold in an execution of the protocol. Once the statement h is generated, we update the knowledge base K with h . The process of updating K with h , denoted $K \oplus h$, is explained below:

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

Definition 5.2. Given a solved deduction statement f , we define the *canonical form* of f to be the statement $f \Downarrow$ obtained by first applying Rule RENAME below as many

times as possible and then applying Rule REMOVE below as many times as possible:

$$\begin{array}{c} \text{RENAME} \frac{H \Leftarrow k_u(X, x), k_{uv}(Y, x), B_1, \dots, B_n}{(H \Leftarrow k_u(X, x), B_1, \dots, B_n)\{Y \mapsto X\}} \\ \text{REMOVE} \frac{H \Leftarrow k_u(X, x), B_1, \dots, B_n \quad x \notin \text{vars}(H)}{H \Leftarrow B_1, \dots, B_n} \end{array}$$

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

It is easy to see that any statement 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. This check ensures that we do not add unnecessary knowledge statements that could otherwise entail non-termination (see Example 5.7 below).

Definition 5.3. The set of *consequences* of a knowledge base K , denoted $\mathbf{conseq}(K)$, is the smallest set such that

$$\begin{array}{c} \text{AXIOM} \frac{}{k_{uv}(R, t) \Leftarrow k_u(R, t), B_1, \dots, B_m \in \mathbf{conseq}(K)} \\ \text{RES} \frac{\begin{array}{c} k_u(R, t) \Leftarrow B_1, \dots, B_n \in K \quad \sigma \text{ a substitution} \\ B_1\sigma \Leftarrow C_1, \dots, C_m \in \mathbf{conseq}(K) \quad \dots \quad B_n\sigma \Leftarrow C_1, \dots, C_m \in \mathbf{conseq}(K) \end{array}}{k_{uv}(R, t)\sigma \Leftarrow C_1, \dots, C_m \in \mathbf{conseq}(K)} \end{array}$$

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 $k_{\ell_1, \dots, \ell_k}(R, t)$. Otherwise, let

$$f\Downarrow = k_{\ell_1, \dots, \ell_k}(R, t) \Leftarrow k_{\ell_1, \dots, \ell_{i_1}}(X_1, t_1), \dots, k_{\ell_1, \dots, \ell_{i_n}}(X_n, t_n)$$

and

$$K \oplus f = \begin{cases} K \cup \{f\Downarrow\} & \text{if } f \text{ is solved and for any } R' \text{ we have that} \\ & k_{\ell_1, \dots, \ell_k}(R', t) \Leftarrow \{k_{\ell_1, \dots, \ell_{i_j}}(X_j, t_j)\}_{j \in \{1, \dots, n\}} \notin K' \\ K \cup \{i_{\ell_1, \dots, \ell_k}(R, R') \Leftarrow \{k_{\ell_1, \dots, \ell_{i_j}}(X_j, t_j)\}_{j \in \{1, \dots, n\}}\} & \text{if } f \text{ is solved and } R' \text{ is such that} \\ & k_{\ell_1, \dots, \ell_k}(R', t) \Leftarrow \{k_{\ell_1, \dots, \ell_{i_j}}(X_j, t_j)\}_{j \in \{1, \dots, n\}} \in K' \\ K \cup \{f\Downarrow\} & \text{if } f \text{ is not solved,} \end{cases}$$

where $K' = \mathbf{conseq}(K_{\text{solved}})$.

Please note that update is not a function, namely that there may be several $R', i_1, \dots, i_n$ such that $k_{\ell_1, \dots, \ell_k}(R', t) \Leftarrow k_{\ell_1, \dots, \ell_{i_1}}(X_1, t_1), \dots, k_{\ell_1, \dots, \ell_{i_n}}(X_n, t_n) \in \mathbf{conseq}(K_{\text{solved}})$. However, we need to compute only one such $R', i_1, \dots, i_n$.

Initial knowledge base. One question that naturally arises is the following: What is the initial knowledge base for the saturation procedure? Given a trace T , the initial knowledge base for the saturation procedure is defined as follows.

Definition 5.4. 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 , that is, $K_i(S) = (((\emptyset \oplus f_1) \oplus f_2) \dots \oplus f_\ell)$ where $f_1, \dots, f_\ell$ is an enumeration of the statements in S . If T is a ground trace, then we write $K_i(T)$ for $K_i(\text{seed}(T))$.

Please observe that $K_i(T)$ depends on the order in which statements in $\text{seed}(T)$ 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 $\text{sat}(K_i(T))$. The reason for choosing $K_i(T)$ instead of $\text{seed}(T)$ as the starting point of the saturation procedure is that $\text{seed}(T)$ may not be a knowledge base (recall that a knowledge base is a set of well-formed statements). For instance, given a trace $T = \text{in}(c, x).\text{out}(c, x)$ we have that $k_{\text{in}(c, x).\text{out}(c)}(w_1, x) \Leftarrow k(X, x) \in \text{seed}(T)$. The set $K_i(T)$ is, however, a knowledge base. This is an immediate consequence of the following proposition.

PROPOSITION 5.5. *If K is a knowledge base and f is a statement, then $K \oplus f$ is a knowledge base.*

PROOF. We first observe that if a statement f is well formed, then $K \oplus f$ is a knowledge base, as equational statements are well formed and the canonical form preserves well formedness, that is, if f is well formed, then $f \Downarrow$ is well formed as well.

If a statement f is not well formed, then it is a solved statement of the form

$$k_u(R, x) \Leftarrow B_1, \dots, B_n.$$

By Definition 4.1, $k_{u'}(X, x) \in B_1, \dots, B_n$ where u' is a prefix of u . By rule AXIOM

$$k_u(X, x) \Leftarrow B_1, \dots, B_n \in \mathbf{conseq}(K)$$

and therefore we have that $K \oplus f = K \cup \{k_u(R, X) \Leftarrow B_1, \dots, B_n\}$, which is a knowledge base as equational statements are well formed. $\square$

Example 5.6. Continuing Example 4.5 on the trace

$$T = \text{in}(c, x).[fst(x) \stackrel{?}{=} a].\text{out}(c, h(s, \text{snd}(x))).\text{out}(c, s),$$

we have that $K_i(\text{seed}(T)) = \text{seed}(T)$. After saturating the initial knowledge base the set $\text{sat}(K_i(T))_{\text{solved}}$ contains in particular the following additional solved statements:

$$k_{\text{in}(c, \text{pair}(a, x)).\text{test}.\text{out}(c)}(w_1, h(s, x)) \Leftarrow k(X, x), \quad (14)$$

$$k_w(w_2, s) \Leftarrow k(X, x), \quad (15)$$

$$r_w \Leftarrow k(X, x), \quad (16)$$

$$i_w(w_1, h(w_2, X_2)) \Leftarrow k(X_1, x), k(X_2, x), k_w(X_3, x), \quad (17)$$

$$ri_w(w_1, h(w_2, X_2)) \Leftarrow k(X_1, x), k(X_2, x), k_w(X_3, x), k(X_1, x), \quad (18)$$

where $w = \text{in}(c, \text{pair}(a, x)).\text{test}.\text{out}(c).\text{out}(c)$.

Statement (14) is obtained by first applying RESOLUTION on statements (5) and (12) (defined in Example 4.5), yielding

$$k_{\text{in}(c, \text{pair}(a, x)).\text{test}.\text{out}(c)}(w_1, h(s, x)) \Leftarrow k(Y, a), k(X, x).$$

Applying again RESOLUTION on the above statement and statement (7), we obtain (14). Statements (15) and (16) are obtained in a similar way.

To obtain statement (17) we apply EQUATION on statements (14) and (13), yielding

$$i_w(w_1, h(Y_1, Y_2)) \Leftarrow k(X, x), k_w(Y_1, s), k_w(Y_2, x).$$

Applying RESOLUTION on this statement and statement (15) yields (17).

Statement (18) is obtained by applying TEST on statements (16) and (17).

Example 5.7. We now present a second, contrived, example that illustrates the need of computing an update based on our set of consequences. Consider the signature $\mathcal{F} = \{f, g, h\}$ where $ar(f) = ar(g) = 2$, $ar(h) = 1$, the (subterm convergent) rewrite rule

$$g(f(f(x_1, h(x_2)), y), f(f(z, h(y)), x_1)) \rightarrow f(z, h(y))$$

and the trace

$$T = \mathbf{out}(c, h(s)).\mathbf{in}(c, x).\mathbf{out}(c, f(x, s)).$$

The initial knowledge base contains the following deduction statements.

$$k_{\mathbf{out}(c)}(w_1, h(s)) \Leftarrow, \quad (19)$$

$$k_{\mathbf{out}(c).\mathbf{in}(c, x).\mathbf{out}(c)}(w_2, f(x, s)) \Leftarrow k_{\mathbf{out}(c)}(X, x), \quad (20)$$

$$k_{w^i}(h(X), h(x)) \Leftarrow k_{w^i}(X, x), \quad (21)$$

$$k_{w^i}(f(X_1, X_2), f(x_1, x_2)) \Leftarrow k_{w^i}(X_1, x_1), k_{w^i}(X_2, x_2), \quad (22)$$

$$k_{w^i}(g(X_1, X_2), g(x_1, x_2)) \Leftarrow k_{w^i}(X_1, x_1), k_{w^i}(X_2, x_2), \quad (23)$$

$$k_{w^i}(g(X_1, X_2), f(z, h(y))) \Leftarrow k_{w^i}(X_1, f(f(x_1, h(x_2)), y)), k_{w^i}(X_2, f(f(z, h(y)), x_1)), \quad (24)$$

for $1 \leq i \leq 3$ where $w = \mathbf{out}(c).\mathbf{in}(c, x).\mathbf{out}(c)$ and w^i denotes the prefix of w of size i . Moreover, we write $w^i(t)$ for $w^i(t)\{x \mapsto t\}$.

Applying RESOLUTION to statement (24, $i = 3$) and (20) we obtain the statement

$$k_{w^3(f(x_1, h(x_2)))}(g(w_2, X_2), f(z, h(s))) \Leftarrow k_{w^1}(X, f(x_1, h(x_2))), \\ k_{w^3(f(x_1, h(x_2)))}(X_2, f(f(z, h(s)), x_1)).$$

Applying 3 times RESOLUTION with statement (22), we obtain

$$k_{w^3(f(x_1, h(x_2)))}(g(w_2, f(f(X_2^{11}, X_2^{12}), X_2^2), f(z, h(s)))) \Leftarrow k_{w^1}(X^1, x_1), k_{w^1}(X^2, h(x_2)), \\ k_{w^3(f(x_1, h(x_2)))}(X_2^{11}, z), \\ k_{w^3(f(x_1, h(x_2)))}(X_2^{12}, h(s)), \\ k_{w^3(f(x_1, h(x_2)))}(X_2^2, x_1).$$

Applying twice RESOLUTION with statement (19), and taking the canonical form, we obtain

$$k_{w^3(f(x_1, h(s)))}(g(w_2, f(f(X_2, w_1), X_1)), f(z, h(s))) \Leftarrow k_{w^1}(X_1, x_1), k_{w^3(f(x_1, h(s)))}(X_2, z). \quad (25)$$

As

$$k_{w^3(f(x_1, h(s)))}(f(X_2, w_1), f(z, h(s))) \Leftarrow k_{w^1}(X_1, x_1), k_{w^3(f(x_1, h(s)))}(X_2, z)$$

is a consequence of the previous solved statements (in particular of Equations (19) and (22)) the update will add the equational statement

$$i_{w^3(f(x_1, h(s)))}(g(w_2, f(f(X_2, w_1), X_1)), f(X_2, w_1)) \Leftarrow k_{w^1}(X_1, x_1), k_{w^3(f(x_1, h(s)))}(X_2, z).$$

Suppose we would have added directly statement (25). In that case, we could again apply RESOLUTION to statements (24) ($i = 3$) and (25), which yields

$$k_{w^3(f(x'_1, h(s)))}(g(g(w_2, f(f(X'_2, w_1), X'_1)), X_2), f(z, h(h(s)))) \Leftarrow k_{w^1}(X'_1, x'_1), \\ k_{w^3(f(x'_1, h(s)))}(X'_2, f(x_1, h(x_2))), \\ k_{w^3(f(x'_1, h(s)))}(X_2, f(f(z, h(h(s))), x_1)).$$

Repeatedly applying RESOLUTION as before, we can obtain the statement

$$k_{w^3(f(x_1, h(s)))}(R, f(z, h(h(s)))) \Leftarrow k_{w^1}(X_1, x_1), k_{w^3(f(x_1, h(s)))}(X_2, z),$$

which is similar to statement (25) but with an additional application of h . (We omit the precise form of R for readability.) We see that we could proceed indefinitely to produce statements of the form

$$k_{w^3(f(x_1, h(s)))}(R^n, f(z, h^n(s))) \Leftarrow k_{w^1}(X_1, x_1), k_{w^3(f(x_1, h(s)))}(X_2, z),$$

demonstrating the need of verifying whether a statement is already a consequence or not.

5.1.1. Soundness and Completeness of the Saturation Procedure. We shall now show that the set of solved statements in $\text{sat}(K_i(T))$ is a sound and complete abstraction of a trace T . We need one more definition that extends $\mathcal{H}(K)$ and allows us to establish that $\text{sat}(K_i(T))$ is a complete abstraction of T .

Definition 5.8. Let K be a set of statements. We define $\mathcal{H}_e(K)$ to be the smallest set of ground terms such that $\mathcal{H}(K) \subseteq \mathcal{H}_e(K)$ and closed under the following rules:

$$\begin{array}{c}
 \text{REFL} \frac{}{i_w(R, R) \in \mathcal{H}_e(K)} \quad \text{SYM} \frac{i_w(R_1, R_2) \in \mathcal{H}_e(K)}{i_w(R_2, R_1) \in \mathcal{H}_e(K)} \\
 \\
 \text{TRAN} \frac{i_w(R_1, R_2) \in \mathcal{H}_e(K) \quad i_w(R_1, R_3) \in \mathcal{H}_e(K)}{i_w(R_2, R_3) \in \mathcal{H}_e(K)} \\
 \\
 \text{CONG} \frac{i_w(R_1, R'_1) \in \mathcal{H}_e(K), \dots, i_w(R_n, R'_n) \in \mathcal{H}_e(K) \quad f \in \mathcal{F}, ar(f) = n}{i_w(f(R_1, \dots, R_n), f(R'_1, \dots, R'_n)) \in \mathcal{H}_e(K)} \\
 \\
 \text{EXTEND} \frac{i_u(R, R') \in \mathcal{H}_e(K)}{i_{uw}(R, R') \in \mathcal{H}_e(K)} \\
 \\
 \text{EQUATIONAL CONSEQUENCE} \frac{k_w(R, t) \in \mathcal{H}(K) \quad i_w(R, R') \in \mathcal{H}_e(K)}{k_w(R', t) \in \mathcal{H}_e(K)}
 \end{array}$$

We have that the set of solved statements produced by the saturation procedure is a sound and complete abstraction of the trace T (see the Electronic Appendix for the proof):

THEOREM 5.9. *Let T be a ground trace and let $K = \text{sat}(K_i(T))$.*

- (Soundness.) *For any $f \in K \cup \mathcal{H}_e(K)$, $T \models f$.*
- (Completeness.) *If $(T, \emptyset) \xrightarrow{L_1, \dots, L_n} (S, \varphi)$, then*
  - (1) $\vdash_{L_1\varphi\downarrow, \dots, L_n\varphi\downarrow} \in \mathcal{H}_e(K_{\text{solved}})$.
  - (2) *If $\varphi \vdash^R t$, then $k_{L_1\varphi\downarrow, \dots, L_n\varphi\downarrow}(R, t\downarrow) \in \mathcal{H}_e(K_{\text{solved}})$.*
  - (3) *If $\varphi \vdash^R t$ and $\varphi \vdash^R t$, then $i_{L_1\varphi\downarrow, \dots, L_n\varphi\downarrow}(R, R') \in \mathcal{H}_e(K_{\text{solved}})$.*

5.1.2. Effectiveness of the Saturation Procedure. We have shown that the set of solved statements in $\text{sat}(K_i(T))$ form a sound and complete abstraction for the trace T . However, the set $\text{sat}(K_i(T))$ may, *a priori*, not be computable for several reasons.

- As the set of public names $\mathcal{M}$ is infinite, the set $\text{seed}(T)$ for a ground trace T is infinite as well.
- For the update rule, we have to check that, given a knowledge base K , a term t , labels $\ell_1, \dots, \ell_k$, indices $1 \leq i_1, \dots, i_n \leq k$, variables $x_1, \dots, x_n \in \mathcal{X}$, and recipe variables $X_1, \dots, X_n \in \mathcal{Y}$, whether

$$\exists R. k_{\ell_1, \dots, \ell_k}(R, t) \Leftarrow k_{\ell_1, \dots, \ell_{i_1}}(X_1, x_1), \dots, k_{\ell_1, \dots, \ell_{i_n}}(X_n, x_n) \in \mathbf{conseq}(K_{\text{solved}}).$$

Furthermore, if the check succeeds, then we have to compute one such R .

- The saturation procedure may itself not terminate even if the initial knowledge base is finite.

We now address each of these three reasons.

First, we show that we only need to consider the saturation of the set $K_i(\text{seed}(T, \mathcal{M}_T))$ where $\mathcal{M}_T$ is the (finite) set of public names occurring in T . The set $\text{sat}(K_i(T))$ can then be computed from the set $\text{sat}(K_i(\text{seed}(T, \mathcal{M}_T)))$ by adding the set of clauses $K_{\mathcal{M},R}^{\text{useless}}$ which is not required for the saturation and is defined as follows.

Definition 5.10. Given a set of public names $M \subseteq \mathcal{M}$ and a set of solved reachability statements R we define

$$K_{\mathcal{M},R}^{\text{useless}} = \{k(m, m) \leftarrow\}_{m \in M} \cup \{i(m, m) \leftarrow\}_{m \in M} \cup \{r_u(m, m) \leftarrow B_1, \dots, B_n \mid m \in M, r_u \leftarrow B_1, \dots, B_n \in R\}.$$

The following is proved in the Electronic Appendix:

LEMMA 5.11. *Let T be a trace and $M_T \subseteq \mathcal{M}$ be the public names occurring in T . Then*

$$\text{sat}(K_i(T)) = \text{sat}(K_i(\text{seed}(T, \mathcal{M}_T))) \cup K_{\mathcal{M},R}^{\text{useless}},$$

where R is the set of solved reachability statements in $\text{sat}(K_i(\text{seed}(T, \mathcal{M}_T)))$.

Since the set $K_i(\text{seed}(T, \mathcal{M}_T))$ is finite, this means that all intermediate knowledge bases in the saturation procedure are finite.

Second, we show that the update step can be computed if we only have a finite number of statements in the knowledge base (see the Electronic Appendix for the proof):

LEMMA 5.12. *Given a finite set of solved statements K , term t , labels $\ell_1, \dots, \ell_k$, indices $1 \leq i_1, \dots, i_n \leq k$, variables $x_1, \dots, x_n \in \mathcal{X}$, and recipe variables $X_1, \dots, X_n \in \mathcal{Y}$, it is decidable if there is an R such that $k_{\ell_1, \dots, \ell_k}(R, t) \leftarrow k_{\ell_1, \dots, \ell_{i_1}}(X_1, x_1), \dots, k_{\ell_1, \dots, \ell_{i_n}}(X_n, x_n) \in \text{conseq}(K_{\text{solved}})$. If the answer to the decision procedure is “Yes,” then we can compute one such R .*

Third, we show that our procedure terminates for the class of subterm convergent rewrite systems (Definition 2.5). It has been shown in Anantharaman et al. [2007] that deducibility is undecidable for convergent optimally reducing rewrite systems. As observed in [Abadi and Fournet 2001], this immediately implies that static equivalence is also undecidable for convergent optimally reducing rewrite systems. As our algorithm would allow us to decide static equivalence as a particular case, we cannot expect a general termination result. However, we prove that the saturation procedure does terminate for the class of subterm convergent rewrite systems (see the Electronic Appendix for the proof):

THEOREM 5.13. *Let T be a ground trace and $S = \text{seed}(T)$. For a subterm convergent rewrite system, the computation of $\text{sat}(K_i(S))$ terminates in a finite number of steps.*

We remark that the saturation is nevertheless sound and complete for the more general class of convergent rewrite systems for which the finite variant property holds. Indeed, the procedure may also terminate on protocols that rely on rewrite systems that are not subterm convergent. This is demonstrated in our case studies when analyzing protocols using blind signatures and trapdoor commitment schemes.

5.2. Algorithm

In this section, we describe an algorithm to decide trace inclusion for determinate processes. In algorithm 1, we describe the checks **REACHABILITY** and **IDENTITY**, which allow us to test whether a trace—represented by the set K of solved statements in the saturated knowledge base associated to this trace—is included in a determinate process P (see the Electronic Appendix for the proof):

THEOREM 5.14. *Let T be a ground trace, P a ground process, and $K = (\text{sat}(K_i(T)))_{\text{solved}}$. We have that*

- If $T \sqsubseteq_{ct} P$, then $\text{REACHABILITY}(K, P)$ and $\text{IDENTITY}(K, P)$ hold.
- If P is determinate and $\text{REACHABILITY}(K, P)$ and $\text{IDENTITY}(K, P)$ hold, then $T \sqsubseteq_{ct} P$.

ALGORITHM 1: Tests for checking $T \sqsubseteq_{ct} P$

Function $\text{Reachability}(K, P)$

Input: A ground process P and a saturated knowledge base K

Output: A Boolean

result $\leftarrow$ true

foreach $r_{l_1, \dots, l_n} \leftarrow \{k_{w_i}(X_i, x_i)\}_{i \in \{1, \dots, m\}} \in K$ **do**

  let $c_1, \dots, c_k$ be fresh public names such that

$\sigma : \text{vars}(l_1, \dots, l_n) \rightarrow \{c_1, \dots, c_k\}$ is a bijection

for $i = 1$ **to** n **do**

if $l_i = \text{in}(d_i, t_i)$ **then**

$M_i \leftarrow \text{in}(d_i, R_i)$ where $k_{l_1\sigma, \dots, l_{i-1}\sigma}(R_i, t_i\sigma) \in \mathcal{H}(K)$

else

$M_i \leftarrow l_i$

      /* $l_i \in \{\text{test}, \text{out}(c) \mid c \in \mathcal{C}\}$ */

  result $\leftarrow$ result $\wedge \exists T', \varphi. (P, \emptyset) \xrightarrow{M_1, \dots, M_n} (T', \varphi)$

return result

Function $\text{Identity}(K, P)$

Input: A ground process P and a saturated knowledge base K

Output: A Boolean

result $\leftarrow$ true

foreach $r_{l_1, \dots, l_n}(R, R') \leftarrow \{k_{w_i}(X_i, x_i)\}_{i \in \{1, \dots, m\}} \in K$ **do**

  let $c_1, \dots, c_k$ be fresh public names such that

$\sigma : \text{vars}(l_1, \dots, l_n) \rightarrow \{c_1, \dots, c_k\}$ is a bijection

for $i = 1$ **to** n **do**

if $l_i = \text{in}(d_i, t_i)$ **then**

$M_i \leftarrow \text{in}(d_i, R_i)$ where $k_{l_1\sigma, \dots, l_{i-1}\sigma}(R_i, t_i\sigma) \in \mathcal{H}(K)$

else

$M_i \leftarrow l_i$

      /* $l_i \in \{\text{test}, \text{out}(c) \mid c \in \mathcal{C}\}$ */

$\omega \leftarrow \{X_1 \mapsto x_1\sigma; \dots; X_m \mapsto x_m\sigma\}$

  result $\leftarrow$ result $\wedge \exists T', \varphi. (P, \emptyset) \xrightarrow{M_1, \dots, M_n} (T', \varphi) \wedge (R\omega = R'\omega)\varphi$

return result

Note that performing the tests requires deciding if, given t , and w , $k_w(R, t) \in \mathcal{H}(K)$ for some recipe R for a knowledge base K containing only solved statements. It is easy to see that this is equivalent to checking if $(k_w(R, t) \Leftarrow) \in \mathbf{conseq}(K)$, and we have already shown that there is an effective procedure for this (which finds an R if such an R exists).

Example 5.15. We continue Example 5.6. Let

$$T = \mathbf{in}(c, x).[\text{fst}(x) \stackrel{?}{=} a].\mathbf{out}(c, h(s, \text{snd}(x))).\mathbf{out}(c, s)$$

and

$$T' = \mathbf{in}(c, x).[\mathbf{fst}(x) \stackrel{?}{=} a].\mathbf{out}(c, h(s, \mathbf{snd}(x))).\mathbf{out}(c, s').$$

The equivalence $T \approx_{ft} T'$ models real-or-random secrecy of s . Our algorithm can be used to show that $T \not\approx_{ft} T'$. In particular, $\text{IDENTITY}(K, P)$ does not hold. Indeed, as shown in Example 5.6, we have that

$$\mathbf{ri}_w(w_1, h(w_2, X_2)) \Leftarrow k(X_1, x), k(X_2, x), k_w(X_3, x), k(X_1, x) \in \text{sat}(K_i(T))_{\text{solved}},$$

where $w = \mathbf{in}(c, \text{pair}(a, x)).\mathbf{test}.\mathbf{out}(c).\mathbf{out}(c)$. Let $\sigma = \{x \mapsto c_1\}$. We have that $k(\text{pair}(a, c_1), \text{pair}(a, c_1)) \in \mathcal{H}(K)$ and

$$(T', \emptyset) \xrightarrow{\mathbf{in}(c, \text{pair}(a, c_1)).\mathbf{test}.\mathbf{out}(c).\mathbf{out}(c)} (\epsilon, \varphi),$$

where $\varphi = \{w_0 \mapsto h(s, c_1), w_1 \mapsto s'\}$. However, $(w_0 \neq h(w_1, c))\varphi$ demonstrating that real-or-random secrecy does not hold.

6. PROTOTYPE AND CASE STUDIES

6.1. The AKiSs Prototype

We implemented the procedure for checking equivalence in a prototype, AKiSs. AKiSs is written in OCaml and has about 2,000 lines of source code, including code for computing complete sets of finite variants and complete sets of equational unifiers. We used AKiSs to verify the equivalences in Examples 3.4 and 3.9. Using AKiSs, we were able to verify strong secrecy for Denning-Sacco-Blanchet [Blanchet 2004] and Needham-Schroeder-Lowe (NSL) [Lowe 1996], resistance to guessing attacks in the Electronic Key Exchange (EKE) protocol [Bellovin and Merritt 1992] and, more interestingly, anonymity of the Fujioka, Okamoto and Ohta (FOO) [Fujioka et al. 1992] and Okamoto [Okamoto 1997] electronic voting protocols.¹ To our knowledge, AKiSs is the only tool that can verify FOO and Okamoto completely automatically. We discuss each of these examples in more detail below. In Arapinis et al. [2013], the tool has also been extended to verify a property called *everlasting privacy* that appears in electronic voting. Several other protocols were analyzed in this context. AKiSs along with all the discussed examples is available on:

<http://akiss.gforge.inria.fr>

In order to ease protocol specification, we add to the process calculus syntax an operator *interleave*, denoted $\parallel$, which models parallel composition of processes, and an operator *sequence*, denoted $;$, which is used in modeling of protocols structured in phases. These constructs are merely syntactic sugar and are defined as follows. Given processes P and Q , we define $P; Q$ as the sequential composition of each trace in P with each trace in Q , that is,

$$P; Q = \{T_1.T_2 \mid T_1 \in P, T_2 \in Q\}.$$

Let ϵ denote the empty sequence, a_1, a_2 be actions and T, T_1, T_2 traces. The parallel composition of two traces is the process defined inductively as

$$\begin{aligned} T \parallel \epsilon &= \epsilon \parallel T = \{T\} \\ a_1.T_1 \parallel a_2.T_2 &= (a_1;(T_1 \parallel a_2.T_2)) \cup (a_2;(a_1.T_1 \parallel T_2)). \end{aligned}$$

The parallel composition is then naturally lifted to process, that is, $P \parallel Q = \bigcup_{T_1 \in P, T_2 \in Q} T_1 \parallel T_2$.

¹Please note that, as defined in Okamoto [1997], modeling of Okamoto's protocol requires private channels. As we do not have private channels in our calculus, we transform the protocol so every message sent by honest participants on a private channel is sent encrypted under a key not known to the adversary

The $\parallel$ operator reflects the usual notion of parallel composition in process calculi. One may note that the number of possible interleavings (and hence generated traces) is exponential. We can, however, slightly lower this number due to the fact that test actions are silent, that is, unobservable. We therefore define an optimized interleaving operator $\parallel_o$ that generates fewer interleavings. In practice, this gain is substantial on several examples. In the following, we let τ (and decorations of τ) range over test actions, that is, actions of the form $[s \stackrel{?}{=} t]$ for some terms s, t . α (and decorations of α) range over input and output actions. The optimized parallel composition of two traces is the process defined inductively as

$$\begin{aligned} \epsilon \parallel_o T &= T \parallel_o \epsilon = \{T\} \\ \tau_1 \dots \tau_n \parallel_o \tau'_1 \dots \tau'_m &= \{\epsilon\} \\ \tau_1 \dots \tau_n.\alpha.T \parallel_o \tau'_1 \dots \tau'_m &= \tau'_1 \dots \tau'_m \parallel_o \tau_1 \dots \tau_n.\alpha.T = \{\tau_1 \dots \tau_n.\alpha.T\} \\ \tau_1 \dots \tau_n.\alpha.T \parallel_o \tau'_1 \dots \tau'_m.\alpha'.T' &= \frac{\tau_1 \dots \tau_n.\alpha; (T \parallel_o \tau'_1 \dots \tau'_m.\alpha'.T') \cup \tau'_1 \dots \tau'_m.\alpha'; (\tau_1 \dots \tau_n.\alpha.T \parallel_o T')}{\tau_1 \dots \tau_n.\alpha.T \parallel_o \tau'_1 \dots \tau'_m.\alpha'.T'} \end{aligned}$$

Intuitively, we consider sequences of silent actions together with the following visible action as atomic. We will now show that this is indeed a sound optimization when checking trace equivalence by showing that $P_1 \parallel P_2 \approx_t P_1 \parallel_o P_2$ (see the Electronic Appendix for a proof).

PROPOSITION 6.1. *Let T_1, T_2 be two ground traces.*

$$\exists S.(T_1 \parallel T_2, \varphi) \xrightarrow{l_1 \dots l_k} (S, \varphi_e) \text{ iff } \exists S'.(T_1 \parallel_o T_2, \varphi) \xrightarrow{l_1 \dots l_k} (S', \varphi_e).$$

From this proposition, it is easy to conclude that $(P \parallel Q) \approx_t (P \parallel_o Q)$.

6.2. Security Properties and Case Studies

We now give more details about our case studies.

Strong flavors of confidentiality. The *strong secrecy* property was introduced in Blanchet [2004] 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:

$$\begin{aligned} A \rightarrow B &: \text{aenc}(\text{sign}(\text{pair}(\text{pk}(ska), \text{pair}(\text{pk}(skb), k))), ska), \text{pk}(skb)) \\ B \rightarrow A &: \text{enc}(x, k). \end{aligned}$$

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 . The detailed protocol model is given in Figure 4. We note that the rewrite system is subterm convergent. 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 that $P_1 \approx_{ct} P_2$. The verification succeeds as expected.

Rewrite System:

$$\begin{array}{lll} \text{fst}(\text{pair}(x, y)) \rightarrow x & \text{adec}(\text{aenc}(x, \text{pk}(y)), y) \rightarrow x & \text{check}(\text{sign}(x, y), \text{pk}(y)) \rightarrow \text{ok} \\ \text{snd}(\text{pair}(x, y)) \rightarrow y & \text{dec}(\text{enc}(x, y), y) \rightarrow x & \text{msg}(\text{sign}(x, y)) \rightarrow x \end{array}$$

Processes:

$$\begin{aligned} P_i &= \text{Setup}; (A \parallel B_i) \quad (i \in \{1, 2\}) \\ \text{Setup} &= \mathbf{out}(c, \text{pk}(skA)).\mathbf{out}(c, \text{pk}(ekB)).\mathbf{in}(c, x_1).\mathbf{in}(c, x_2) \\ A &= \mathbf{out}(c, \text{aenc}(\text{sign}(\text{pair}(\text{pk}(skA), \text{pair}(\text{pk}(ekB), k)), skA), \text{pk}(ekB))) \\ B_i &= \mathbf{in}(c, z). \\ &\quad [\text{check}(\text{adec}(z, ekB), \text{pk}(skA)) \stackrel{?}{=} \text{ok}]. \\ &\quad [\text{fst}(\text{msg}(\text{adec}(z, ekB))) \stackrel{?}{=} \text{pk}(skA)]. \\ &\quad [\text{fst}(\text{snd}(\text{msg}(\text{adec}(z, ekB)))) \stackrel{?}{=} \text{pk}(ekB)]. \\ &\quad \mathbf{out}(c, \text{enc}(x_i, \text{snd}(\text{snd}(\text{msg}(\text{adec}(z, ekB))))) \end{aligned}$$

Fig. 4. Formal description of the protocol by Blanchet.

Rewrite System:

$$\begin{array}{lll} \text{fst}(\text{pair}(x, y)) \rightarrow x & \text{adec}(\text{aenc}(x, \text{pk}(y)), y) \rightarrow x & \text{check}(\text{sign}(x, y), \text{pk}(y)) \rightarrow \text{ok} \\ \text{snd}(\text{pair}(x, y)) \rightarrow y & \text{dec}(\text{enc}(x, y), y) \rightarrow x & \text{msg}(\text{sign}(x, y)) \rightarrow x \end{array}$$

Processes:

$$\begin{aligned} P_i &= \text{Setup}; (A_i \parallel B_i) \quad (i \in \{1, 2\}) \\ \text{Setup} &= \mathbf{out}(c, \text{pk}(skA)).\mathbf{out}(c, \text{pk}(ekB)).\mathbf{in}(c, x_1).\mathbf{in}(c, x_2).\mathbf{in}(c, y_1).\mathbf{in}(c, y_2) \\ A_i &= \mathbf{out}(c, \text{pair}(\text{aenc}(\text{sign}(\text{pair}(\text{pk}(skA), \text{pair}(\text{pk}(ekB), k)), skA), \text{pk}(ekB))), \\ &\quad \text{enc}(y_1, k)) \\ B_i &= \mathbf{in}(c, z). \\ &\quad [\text{check}(\text{adec}(z, ekB), \text{pk}(skA)) \stackrel{?}{=} \text{ok}]. \\ &\quad [\text{fst}(\text{msg}(\text{adec}(z, ekB))) \stackrel{?}{=} \text{pk}(skA)]. \\ &\quad [\text{fst}(\text{snd}(\text{msg}(\text{adec}(z, ekB)))) \stackrel{?}{=} \text{pk}(ekB)]. \\ &\quad \mathbf{out}(c, \text{enc}(x_i, \text{snd}(\text{snd}(\text{msg}(\text{adec}(z, ekB))))) \end{aligned}$$

Fig. 5. Formal description of the variant protocol by Blanchet.

A variant of the protocol [Blanchet 2004] consists in letting A also send out a secret y encrypted with k changing the first message to

$$A \rightarrow B : \text{pair}(\text{aenc}(\text{sign}(\text{pair}(\text{pk}(ska), \text{pair}(\text{pk}(skb), k))), ska), \text{pk}(skb)), \text{enc}(y, k)).$$

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 . The detailed model is given in Figure 5. 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) [Lowe 1996] protocol. The NSL protocol is a two-

Rewrite System:

$$\text{fst}(\text{pair}(x, y)) \rightarrow x \quad \text{snd}(\text{pair}(x, y)) \rightarrow y \quad \text{adec}(\text{aenc}(x, \text{pk}(y), z), y) \rightarrow x$$

Processes:

$$\begin{aligned} P_i &= \text{Setup}; (A \parallel B_i) \quad (i \in \{1, 2\}) \\ \text{Setup} &= \mathbf{out}(c, \text{pk}(skA)).\mathbf{out}(c, \text{pk}(ekB)).\mathbf{out}(c, skD).\mathbf{in}(c, x_1).\mathbf{in}(c, x_2) \\ A &= \mathbf{out}(c, \text{aenc}(\text{pair}(n_a, a), \text{pk}(skD), r_1)). \\ &\quad \mathbf{in}(c, y).[\text{snd}(\text{snd}(\text{adec}(y, skA))) = d]. \\ &\quad \mathbf{out}(c, \text{aenc}(\text{fst}(\text{snd}(\text{adec}(y, skA))), \text{pk}(skD), r_2)) \\ B_i &= \mathbf{in}(c, z).[\text{snd}(\text{adec}(z, skB)) = a]. \\ &\quad \mathbf{out}(c, \text{aenc}(\text{pair}(\text{fst}(\text{adec}(z, skB)), \text{pair}(x_1, b)), \text{pk}(skA), r_3)) \end{aligned}$$

Fig. 6. Formal description of the NSL protocol.

way handshake protocol relying only on public encryption of fresh nonces and can be informally described as follows:

$$\begin{aligned} A \rightarrow B &: \text{aenc}(\text{pair}(n_a, A), \text{pk}(skb)) \\ B \rightarrow A &: \text{aenc}(\text{pair}(n_b, \text{pair}(n_a, B)), \text{pk}(ska)) \\ A \rightarrow B &: \text{aenc}(\text{pair}(n_b), \text{pk}(skb)). \end{aligned}$$

Once again, the modelling of NSL leads to a subterm convergent rewrite system and determinate processes. We therefore used $\approx_{ct}$ for our verification. The detailed model is given in Figure 6.

This model includes a session of the initiator who is willing to engage with any participant (including the attacker to allow man-in-the-middle attacks) and a session of B who is willing to engage a session with A. Note that if B was willing to start a session with an arbitrary initiator, then the secrecy of n_b would be trivially broken in a session with the attacker. (In a more complex model, one could of course add additional sessions for B with an arbitrary initiator.) We note that for the verification of NSL, one needs to explicitly model randomness for asymmetric encryption since the protocol is insecure if deterministic asymmetric encryption is used. Indeed, as the attacker may choose the value of n_b , he or she could simply recompute the last message and compare it with the message sent by the initiator.

We also used AKiSs to verify the above protocols for *real-or-random* secrecy. Let P be a protocol and $n \in \text{names}(P)$. Then n is said to be *real-or-random secret* if

$$P; \mathbf{out}(c, n) \approx_t P; \mathbf{out}(c, n'),$$

where n' is a fresh name, that is, a name that does not appear in P . Real-or-random secrecy is particularly useful to model resistance to offline guessing attacks in password protocols [Baudet 2005]. Intuitively, an offline guessing attacks works in two phases. In the first (online) phase, the attacker interacts with the protocol P in an arbitrary way. In a second (offline) phase, the attacker tries all possible passwords against the data recorded in the first phase. Our property states that the attacker cannot distinguish the case where he or she tests the real password (n) from the case where he or she tests a wrong password (n'). We show that the EKE protocol [Bellare and Merritt 1992] is resistant to offline guessing attacks. The protocol can be described informally as

Rewrite System:

$$\begin{array}{ll} \text{fst}(\text{pair}(x, y)) \rightarrow x & \text{adec}(\text{aenc}(x, \text{pk}(y), z), y) \rightarrow x \\ \text{snd}(\text{pair}(x, y)) \rightarrow y & \text{dec}(\text{enc}(x, y), y) \rightarrow x \end{array}$$

Processes:

$$\begin{array}{l} P_1 = (A \parallel B); \mathbf{out}(c, w) \\ P_2 = (A \parallel B); \mathbf{out}(c, w') \\ A = \mathbf{out}(c, \text{enc}(\text{pk}(k), w)). \\ \quad \mathbf{in}(c, x_1). \\ \quad \mathbf{out}(c, \text{enc}(n_a, \text{adec}(\text{dec}(x_1, w), k))). \\ \quad \mathbf{in}(c, x_2). [\text{fst}(\text{dec}(x_2, \text{adec}(\text{dec}(x_1, w), k))) \stackrel{?}{=} n_a]. \\ \quad \mathbf{out}(c, \text{snd}(\text{dec}(x_2, \text{adec}(\text{dec}(x_1, w), k)))). \\ B = \mathbf{in}(c, y_1). \\ \quad \mathbf{out}(c, \text{enc}(\text{aenc}(r, \text{dec}(y_1, w), r_b), w)). \\ \quad \mathbf{in}(c, y_2). \\ \quad \mathbf{out}(c, \text{enc}(\text{pair}(\text{dec}(y_2, r), n_b), r)). \end{array}$$

Fig. 7. Formal description of the EKE protocol.

follows:

$$\begin{array}{l} A \rightarrow B : \text{enc}(\text{pk}(k), w) \\ B \rightarrow A : \text{enc}(\text{aenc}(r, \text{pk}(k)), w) \\ A \rightarrow B : \text{enc}(n_a, r) \\ B \rightarrow A : \text{enc}(\langle n_a, n_b \rangle, r) \\ A \rightarrow B : \text{enc}(n_b, r). \end{array}$$

In the first step, A generates a new private session key k and sends the corresponding public key $\text{pk}(k)$ to B, encrypted (using symmetric encryption) with the shared password w . Then, B generates a fresh symmetric session key r , which he or she encrypts (using asymmetric encryption) with the previously received public key $\text{pk}(k)$. Finally, he or she encrypts the resulting ciphertext with the password w and sends the result to A. The last three steps perform a handshake to avoid replay attacks. Using AKiSs, we have shown that the protocol resists offline guessing attacks on the password w . As EKE is modelled by a subterm convergent rewrite system and determinate processes, we used the $\approx_{ct}$ relation. The detailed description of our model is given in Figure 7.

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 [Fujioka et al. 1992] and the Okamoto protocol [Okamoto 1997]. Voter privacy is naturally modelled as an equivalence property [Delaune et al. 2009b; Backes et al. 2008]: 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 modelling of the protocols, which we make precise below, is exactly the same as in Delaune et al. [2009b]. 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 modelled directly.

We now briefly describe the two protocols. The FOO protocol relies on blind signatures and a commitment function. The rewrite system is specified in Figure 8. We note that the rewrite system is not subterm convergent, but it is optimally reducing. The protocol

Rewrite System:

$$\begin{aligned} \text{open}(\text{commit}(x, y), y) &\rightarrow x & \text{check}(\text{sign}(x, y), \text{pk}(y)) &\rightarrow x \\ \text{unblind}(\text{sign}(\text{blind}(x, y), z), y) &\rightarrow \text{sign}(x, z) \end{aligned}$$

Processes:

$$\begin{aligned} P_1 &= \text{Setup}; A_{yes}B_{no} \\ P_2 &= \text{Setup}; A_{no}B_{yes} \\ \text{Setup} &= \mathbf{out}(c, \text{pk}(sk_A)).\mathbf{out}(c, \text{pk}(sk_B)) \\ V\text{-Phase1} &= \mathbf{out}(V, \text{sign}(\text{blind}(\text{commit}(v, r), b), sk)). \\ &\quad \mathbf{in}(A, x).[\text{check}(x, \text{pk}(sk_A)) \stackrel{?}{=} \text{blind}(\text{commit}(v, r), b)] \\ V\text{-Phase2} &= \mathbf{out}(C, \text{unblind}(x, b)) \\ V\text{-Phase3} &= \mathbf{out}(C, r) \\ AB &= (V\text{-Phase1} \{v_a/v, sk_A/sk, r_a/r, b_a/b, x_a/x\} \parallel \\ &\quad V\text{-Phase1} \{v_b/v, sk_B/sk, r_b/r, b_b/b, x_b/x\}); \\ &\quad (V\text{-Phase2} \{b_a/b, x_a/x\} \parallel V\text{-Phase2} \{b_b/b, x_b/x\}); \\ &\quad (V\text{-Phase3} \{r_a/r\} \parallel V\text{-Phase3} \{r_b/r\}) \\ A_{yes}B_{no} &= AB \{^{yes}/v_a, ^{no}/v_b\} \\ A_{no}B_{yes} &= AB \{^{no}/v_a, ^{yes}/v_b\} \end{aligned}$$

Fig. 8. Formal description of the FOO protocol.

consists of three phases informally described as follows:

Phase 1 :
 $V \rightarrow A : \text{sign}(\text{blind}(\text{commit}(v, r), b), sk_V)$
 $A \rightarrow V : \text{sign}(\text{blind}(\text{commit}(v, r), b), sk_A)$
Phase 2 :
 $V \rightarrow C : \text{sign}(\text{commit}(v, r), sk_A)$
Phase 3 :
 $V \rightarrow C : r.$

In the first phase, the voter V commits to his or her vote v , which he or she blindly signs and sends to the election administrator A . A checks eligibility of V and then signs the blinded commitment. Blinding the commitment ensures that A cannot trace the ballot. V unblinds the signature and obtains a ballot that is signed by A . In the second phase, V submits the signed ballot to a collector C , who publishes all the submitted ballots on a public bulletin board. Finally, in the third phase, V submits the random r , which allows us to open the commitment to C , who again publishes this value on the bulletin board. The election can now be tallied by any observer. The detailed model is given in Figure 8. Note that only two honest voters need to be modelled for showing anonymity. All remaining voters and election authorities are subsumed by the adversary. The processes $A_{yes}B_{no}$ and $A_{no}B_{yes}$ model the situation where these two honest voters have swapped their vote. The protocols do not lead to determinate processes. Therefore, we proved the relation $A_{yes}B_{no} \approx_{ft} A_{no}B_{yes}$.

We will not give a detailed description of the Okamoto protocol and refer the reader to Delaune et al. [2009b]. The protocol is a variant of the FOO protocol, which aims to achieve receipt freeness. To avoid vote selling, a voter should not be able to provide a receipt of how he or she voted to a potential coercer. In the FOO protocol, this is possible by sending all private names to a coercer. The main tool to avoid this problem

in the Okamoto protocol is the use of trapdoor commitment functions. These functions allow us to change the value of committed vote using a secret value called the trapdoor. Following Ciobăcă et al. [2009], we model trapdoor commitment by the following rewrite system:

$$\begin{array}{ll} \text{open}(\text{tdcommit}(x, y, z), y) \rightarrow x & \text{tdcommit}(x, f(x_1, y, z, x), z) \rightarrow \text{tdcommit}(x_1, y, z) \\ \text{open}(\text{tdcommit}(x, y, z), f(x, y, z, x_1)) \rightarrow x_1 & f(x_1, f(x, y, z, x_1), z, x_2) \rightarrow f(x, y, z, x_2). \end{array}$$

Intuitively, a trapdoor commitment $\text{tdcommit}(x, y, z)$ commits to x using the key y and trapdoor z . The commitment can be opened using key y to x . However, knowing the trapdoor z , one may compute an alternate key $f(x_1, y, z, x)$ that opens the commitment $\text{tdcommit}(x, y, z)$ to x_1 rather than x . This rewrite system is again optimally reducing but not subterm convergent and out of the scope of most tools, even in the simpler case of a passive adversary. The only result we are aware of that can verify protocols for the case of passive adversary and that uses trapdoor commitments is Ciobăcă et al. [2009]. As for the FOO protocol, we used the relation $\approx_{\bar{R}}$ to prove anonymity.

To our knowledge, no other tool can handle the above two protocols automatically. We are aware of two other attempts for verifying the FOO protocol. Using ProVerif [Blanchet 2004], Delaune et al. [2008] verify a transformation of the protocol. However, the soundness of this transformation has never been proven. Chothia et al. [2007] 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 only a finite state system.

Efficiency. On a standard modern laptop, AKiSs takes a few seconds to carry out the above verification, except for the verification of the Okamoto protocol, which takes about 30s. 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, we believe that we can scale to larger protocols and more sessions by parallelizing the saturation of these traces (e.g., on clusters of machines). An implementation performing saturations in parallel is currently in progress.

7. CONCLUSION

In this article we present a novel procedure for verifying equivalence properties for a bounded number of sessions of cryptographic protocols. The procedure has been implemented in a tool that is able to handle examples that are out of the scope of existing tools.

There are several directions for future work. The implementation of the tool should be optimized, and we plan to analyze more examples coming from electronic voting, RFID protocols, and auction protocols, which all have requirements stated in terms of equivalences.

We would also like to extend the procedure to be able to take disequalities into account. On the one hand, disequalities will allow us to verify processes with else branches, which are important in a number of practical examples. On the other hand, characterizing disequalities in our decision procedure would allow us to directly decide trace equivalence based on static equivalence (rather than static inclusion). Another direction would be to extend the procedure to allow Associative-Commutative operators in order to treat protocols based on exclusive-or and Diffie-Hellman exponentiations.

ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library.

ACKNOWLEDGMENTS

We thank David Baelde, Stéphanie Delaune, and Ivan Gazeau for interesting discussions and comments on previous versions of this article. We also thank the anonymous reviewers for their detailed comments.

REFERENCES

2015. French expats vote online in 2012 legislative elections. (2015). http://www.parliament.uk/documents/speaker/digital-democracy/FR_Successcase.pdf.
2015. Statistics about Internet Voting in Estonia. (2015). <http://www.vvk.ee/voting-methods-in-estonia/engindex/statistics>.
- Martín Abadi and Véronique Cortier. 2006. Deciding knowledge in security protocols under equational theories. *Theor. Comput. Sci.* 387, 1–2 (November 2006), 2–32.
- Martín Abadi and Cédric Fournet. 2001. Mobile values, new names, and secure communication. In *Proc. 28th ACM Symposium on Principles of Programming Languages (POPL01)*, Hanne Riis Nielson (Ed.). ACM, London, UK, 104–115.
- Martín Abadi and Cédric Fournet. 2004. Private authentication. *Theor. Comput. Sci.* 322, 3 (2004), 427–476.
- Martín Abadi and Andrew D. Gordon. 1999. A calculus for cryptographic protocols: The spi calculus. *Inf. Comput.* 148, 1 (1999), 1–70.
- Reynald Affeldt and Hubert Comon-Lundh. 2009. Verification of security protocols with a bounded number of sessions based on resolution for rigid variables. In *Formal to Practical Security*. LNCS, State-of-the-Art Survey, Vol. 5458. Springer, 1–20.
- Siva Anantharaman, Paliath Narendran, and Michaël Rusinowitch. 2007. Intruders with caps. In *Proc. 18th International Conference on Term Rewriting and Applications (RTA07) (LNCS)*, Vol. 4533. Springer, 20–35.
- Myrto Arapinis, Tom Chothia, Eike Ritter, and Mark D. Ryan. 2010. Analysing unlinkability and anonymity using the applied pi calculus. In *Proc. 23rd Computer Security Foundations Symposium (CSF10)*. IEEE Comp. Soc. Press, 107–121.
- Myrto Arapinis, Véronique Cortier, Steve Kremer, and Mark D. Ryan. 2013. Practical everlasting privacy. In *Proc. 2nd Conference on Principles of Security and Trust (POST13) (Lecture Notes in Computer Science)*, Vol. 7796. Springer, 21–40.
- Alessandro Armando, David A. Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, Paul Hankes Drielsma, Pierre-Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, Sebastian Mödersheim, David von Oheimb, Michaël Rusinowitch, Judson Santiago, Mathieu Turuani, Luca Viganò, and Laurent Vigneron. 2005. The AVISPA tool for the automated validation of internet security protocols and applications. In *Proc. 17th International Conference on Computer Aided Verification (CAV05) (Lecture Notes in Computer Science)*. Springer, 281–285.
- Mathilde Arnaud, Véronique Cortier, and Stéphanie Delaune. 2007. Combining algorithms for deciding knowledge in security protocols. In *Proc. 6th International Symposium on Frontiers of Combining Systems (FroCoS07) (Lecture Notes in Artificial Intelligence)*, Vol. 4720. Springer, 103–117.
- Franz Baader and Tobias Nipkow. 1998. *Term Rewriting and All That*. Cambridge University Press.
- Franz Baader and Wayne Snyder. 2001. Unification theory. In *Handbook of Automated Reasoning, Volume I, Chapter 8*. Elsevier Science, 445–532.
- Michael Backes, Catalin Hritcu, and Matteo Maffei. 2008. Automated verification of remote electronic voting protocols in the applied pi-calculus. In *Proc. 21st IEEE Computer Security Foundations Symposium (CSF08)*. 195–209.
- Mathieu Baudet. 2005. Deciding security of protocols against off-line guessing attacks. In *12th ACM Conference on Computer and Communications Security (CCS05)*. ACM Press, Alexandria, Virginia, USA, 16–25. DOI : <http://dx.doi.org/10.1145/1102125>
- Mathieu Baudet, Véronique Cortier, and Stéphanie Delaune. 2009. YAPA: A generic tool for computing intruder knowledge. In *Proc. 20th International Conference on Rewriting Techniques and Applications (RTA09) (Lecture Notes in Computer Science)*, Vol. 5595. Springer, 148–163.
- Steven M. Bellovin and Michael Merritt. 1992. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Symposium on Security and Privacy (S&P92)*. IEEE Comp. Soc., Washington, DC, USA, 72–84.
- Bruno Blanchet. 2001. An efficient cryptographic protocol verifier based on prolog rules. In *14th Computer Security Foundations Workshop (CSFW01)*. IEEE Comp. Soc. Press, Cape Breton, Nova Scotia, Canada, 82–96.

- Bruno Blanchet. 2004. Automatic proof of strong secrecy for security protocols. In *Symposium on Security and Privacy (S&P'04)*. 86–100.
- Bruno Blanchet, Martín Abadi, and Cédric Fournet. 2005. Automated verification of selected equivalences for security protocols. In *Symposium on Logic in Computer Science*. IEEE Comp. Soc. Press, Chicago, IL, 331–340.
- Johannes Borgström. 2008. *Equivalences and Calculi for Formal Verification of Cryptographic Protocols*. PhD thesis. EPFL, Switzerland.
- Johannes Borgström, Sébastien Briaies, and Uwe Nestmann. 2004. Symbolic bisimulation in the spi calculus. In *Proc. 15th Int. Conference on Concurrency Theory (LNCS)*, Vol. 3170. Springer, 161–176.
- Mayla Bruso, Konstantinos Chatzikokolakis, and Jerry den Hartog. 2010. Analysing unlinkability and anonymity using the applied pi calculus. In *Proc. 23rd Computer Security Foundations Symposium (CSF'10)*. IEEE Comp. Soc. Press, 107–121.
- Rohit Chadha, Ștefan Ciobăcă, and Steve Kremer. 2012. Automated verification of equivalence properties of cryptographic protocols. In *21st European Symposium on Programming, ESOP 2012 (Lecture Notes in Computer Science)*, Helmut Seidl (Ed.), Vol. 7211. Springer, 108–127.
- Vincent Cheval and Bruno Blanchet. 2013. Proving more observational equivalences with proverif. In *Principles of Security and Trust—Second International Conference, POST*. 226–246.
- Vincent Cheval, Hubert Comon-Lundh, and Stephanie Delaune. 2010. Automating security analysis: Symbolic equivalence of constraint systems. In *Proc. International Joint Conference on Automated Reasoning (IJCAR'10) (Lecture Notes in Artificial Intelligence)*. Springer, 412–426.
- Vincent Cheval, Hubert Comon-Lundh, and Stéphanie Delaune. 2011. Trace equivalence decision: Negative tests and non-determinism. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS'11)*. ACM Press, Chicago, Illinois, USA, 321–330.
- Yannick Chevalier and Michaël Rusinowitch. 2010. Decidability of equivalence of symbolic derivations. *Journal of Automated Reasoning* 48, 2 (2010), 263–292.
- Andrew Cholewa, José Meseguer, and Santiago Escobar. 2014. *Variants of Variants and the Finite Variant Property*. Research report. University of Illinois at Urbana-Champaign. 13 pages. <http://hdl.handle.net/2142/47117>.
- Tom Chothia, Simona Orzan, Jun Pang, and Muhammad Torabi Dashti. 2007. A framework for automatically checking anonymity with μ CRL. In *2nd Symposium on Trustworthy Global Computing (TGC'06) (Lecture Notes in Computer Science)*, Vol. 4661. Springer, 301–318.
- Ștefan Ciobăcă. 2011. *Computing finite variants for subterm convergent rewrite systems*. Research Report LSV-11-06. Laboratoire Spécification et Vérification, ENS Cachan, France. 16 pages. http://www.lsv.ens-cachan.fr/Publis/RAPPORTS_LSV/PDF/rr-lsv-2011-06.pdf.
- Ștefan Ciobăcă, Stéphanie Delaune, and Steve Kremer. 2009. Computing knowledge in security protocols under convergent equational theories. In *Proc. 22nd International Conference on Automated Deduction (CADE'09) (Lecture Notes in Artificial Intelligence)*, Renate Schmidt (Ed.). Springer, Montreal, Canada, 355–370.
- Ștefan Ciobăcă, Stéphanie Delaune, and Steve Kremer. 2011. Computing knowledge in security protocols under convergent equational theories. *J. Automat. Reason.* 48, 2 (2011), 219–262.
- Hubert Comon-Lundh and Stéphanie Delaune. 2005. The finite variant property: How to get rid of some algebraic properties. In *Proceedings of the 16th International Conference on Rewriting Techniques and Applications (RTA'05) (Lecture Notes in Computer Science)*, Vol. 3467. Springer, 294–307.
- Véronique Cortier and Stéphanie Delaune. 2007. Deciding knowledge in security protocols for monoidal equational theories. In *Proc. 14th Int. Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'07) (LNAI)*, Vol. 4790. Springer, Yerevan, Armenia, 196–210. <http://www.loria.fr/~cortier/Papiers/LPAR07.pdf>.
- Véronique Cortier and Stéphanie Delaune. 2009. A method for proving observational equivalence. In *Proc. 22nd IEEE Computer Security Foundations Symposium (CSF'09)*. IEEE Computer Society Press, Port Jefferson, NY, 266–276. DOI: <http://dx.doi.org/10.1109/CSF.2009.9>
- Cas J. F. Cremers. 2008. The scyther tool: Verification, falsification, and analysis of security protocols. In *Proc. 20th International Conference on Computer Aided Verification (CAV'08) (Lecture Notes in Computer Science)*, Vol. 5123. Springer, 414–418.
- Morten Dahl, Stéphanie Delaune, and Graham Steel. 2010. Formal analysis of privacy for vehicular mix-zones. In *Proc. 15th European Symposium on Research in Computer Security (ESORICS'10) (Lecture Notes in Computer Science)*, Vol. 6345. Springer, 55–70.
- Morten Dahl, Stéphanie Delaune, and Graham Steel. 2011. Formal analysis of privacy for anonymous location based services. In *Proc. Workshop on Theory of Security and Applications (TOSCA'11)*. pp 98–112. To appear.

- Stéphanie Delaune, Steve Kremer, and Olivier Pereira. 2009a. Simulation based security in the applied pi calculus. In *Proceedings of the 29th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'09) (Leibniz International Proceedings in Informatics)*, Ravi Kannan and K. Narayan Kumar (Eds.), Vol. 4. Leibniz-Zentrum für Informatik, Kanpur, India, 169–180. DOI : <http://dx.doi.org/10.4230/LIPIcs.FSTTCS.2009.2316>
- Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. 2009b. Verifying privacy-type properties of electronic voting protocols. *J. Comput. Sec.* 17, 4 (July 2009), 435–487. DOI : <http://dx.doi.org/10.3233/JCS-2009-0340>
- Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. 2010. Symbolic bisimulation for the applied pi calculus. *J. Comput. Sec.* 18, 2 (2010), 317–377. <http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/DKR-jcs09.pdf>.
- Stéphanie Delaune, Mark D. Ryan, and Ben Smyth. 2008. Automatic verification of privacy properties in the applied pi-calculus. In *Proceedings of the 2nd Joint iTrust and PST Conferences on Privacy, Trust Management and Security (IFIPTM'08) (IFIP Conference Proceedings)*, Yuecel Karabulut, John Mitchell, Peter Herrmann, and Christian Damsgaard Jensen (Eds.), Vol. 263. Springer, Trondheim, Norway, 263–278. <http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/DRS-ifiptm08.pdf>.
- Danny Dolev and Andrew Chi-Chih Yao. 1981. On the security of public key protocols. In *Proc. of the 22nd Symp. on Foundations of Computer Science*. IEEE Comp. Soc. Press, 350–357.
- Luca Durante, Riccardo Sisto, and Adriano Valenzano. 2003. Automatic testing equivalence verification of spi calculus specifications. *ACM Trans. Softw. Eng. Method.* 12, 2 (2003), 222–284.
- Santiago Escobar, Catherine Meadows, and José Meseguer. 2009. Maude-NPA: Cryptographic protocol analysis modulo equational properties. In *Foundations of Security Analysis and Design V (Lecture Notes in Computer Science)*, Vol. 5705. Springer, 1–50.
- Santiago Escobar, Ralf Sasse, and José Meseguer. 2012. Folding variant narrowing and optimal variant termination. *J. Logic Algebr. Program.* 81, 7–8 (2012), 898–928.
- Atsushi Fujioka, Tatsuaki Okamoto, and Kazui Ohta. 1992. A practical secret voting scheme for large scale elections. In *Advances in Cryptology—AUSCRYPT'92 (Lecture Notes in Computer Science)*, Vol. 718. Springer, 244–251.
- Jean Goubault-Larrecq. 2005. Deciding $\mathcal{H}_1$ by Resolution. *Inform. Process. Lett.* 95, 3 (Aug. 2005), 401–408.
- J. Alex Halderman and Vanessa Teague. 2015. The new south wales ivote system: Security failures and verification flaws in a live online election. *CoRR* abs/1504.05646 (2015). <http://arxiv.org/abs/1504.05646>
- Hans Hüttel. 2002. Deciding framed bisimilarity. In *Proc. 4th International Workshop on Verification of Infinite-State Systems (INFINITY'02)*. 1–20.
- Steve Kremer and Mark D. Ryan. 2005. Analysis of an electronic voting protocol in the applied pi-calculus. In *14th European Symposium on Programming (ESOP'05) (LNCS)*, Mooly Sagiv (Ed.), Vol. 3444. Springer, Edinburgh, U.K., 186–200. DOI : <http://dx.doi.org/10.1007/b107380>
- Jia Liu and Huimin Lin. 2010. A complete symbolic bisimulation for full applied pi calculus. In *Proc. 36th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'10) (Lecture Notes in Computer Science)*, Vol. 5901. Springer, 552–563.
- Gavin Lowe. 1996. Breaking and fixing the needham-schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96) (LNCS)*, T. Margaria and B. Steffen (Eds.), Vol. 1055. Springer-Verlag, 147–166.
- Paliath Narendran, Frank Pfenning, and Richard Statman. 1997. On the unification problem for cartesian closed categories. *J. Symb. Log.* 62, 2 (1997), 636–647.
- Tatsuaki Okamoto. 1997. Receipt-free electronic voting schemes for large scale elections. In *Proc. 5th Int. Security Protocols Workshop (Lecture Notes in Computer Science)*, Vol. 1361. Springer, Paris, France, 25–35.
- Sonia Santiago, Santiago Escobar, Catherine Meadows, and José Meseguer. 2014. A formal definition of protocol indistinguishability and its verification using maude-NPA. In *Proc. 10th International Workshop on Security and Trust Management (STM'14) (Lecture Notes in Computer Science)*, Vol. 8743. Springer, 162–177.
- Alwen Tiu and Jeremy Dawson. 2010. Automating open bisimulation checking for the spi-calculus. In *Proc. 23rd Computer Security Foundations Symposium (CSF'10)*. IEEE Comp. Soc. Press, 307–321.
- Christoph Weidenbach. 1999. Towards an automatic analysis of security protocols in first-order logic. In *Proc. 16th International Conference on Automated Deduction (CADE'99) (Lecture Notes in Computer Science)*, Vol. 1632. Springer, 314–328.

Received March 2015; revised April 2016; accepted April 2016