SAT-Equiv: an efficient tool for equivalence properties

Véronique Cortier*, Antoine Dallon*†‡, Stéphanie Delaune‡
* CNRS, LORIA, France
† LSV & ENS Paris-Saclay, Université Paris-Saclay, France
‡ CNRS, IRISA, France

***Abstract*—Automatic tools based on symbolic models have been successful in analyzing security protocols. Such tools are particularly adapted for trace properties (e.g. secrecy or authentication), while they often fail to analyse equivalence properties.**

Equivalence properties can express a variety of security properties, including in particular privacy properties (vote privacy, anonymity, untraceability). Several decision procedures have already been proposed but the resulting tools are rather inefficient.

In this paper, we propose a novel algorithm, based on graph planning and SAT-solving, which significantly improves the efficiency of the analysis of equivalence properties. The resulting implementation, SAT-Equiv, can analyze several sessions where most tools have to stop after one or two sessions.

I. Introduction

Formal methods have produced several successful tools for the automatic analysis of security protocols. Examples of such tools are ProVerif [1], Avantssar [2], Maude-NPA [3], Scyther [4], or Tamarin [5]. They have been applied to many protocols of the literature including e.g. Kerberos and TLS. However, one type of properties still resists to these tools, namely privacy properties. Privacy properties incude ballot privacy (no one should know how I voted), privacy (no one should know I am here), or unlinkability (no one should be able to relate two of my transactions). Such properties are typically expressed as equivalences: an attacker should not be able to distinguish a session from Alice from one from Bob.

Equivalence properties are harder to analyze than the more standard authentication or confidentiality properties (expressed as trace properties). Among the tools mentioned earlier, only ProVerif, Maude-NPA and Tamarin may handle equivalences. Tamarin often requires user interaction for equivalence properties. Maude-NPA [6] often does not terminate when used for equivalence properties. Since checking equivalence properties for an unbounded number of sessions is undecidable [7], ProVerif may of course also fail. This is in particular the case when the order of the protocol rules matters or when some step may be executed at most once.

The alternative is to *decide* equivalence, for a bounded number of sessions. Several procedures have been proposed [8]–[10], often with a companion tool: Akiss [10], Spec [9], Apte [11]. Unfortunately, these tools have a very limited practical impact because they scale badly. Analyzing one session typically requires several seconds and the analysis of two sessions is often unreachable, although the tools Apte and Akiss have recently improved their efficiency through the use of Partial Order Reduction (POR) techniques [12], [13]. It is interesting to note that considering one or two sessions is not sufficient to explore all standard attack scenarios (where each participant may engage a session with an honest or a dishonest agent and may be involved in any role). For example, in the case of a three-party protocol, with a trusted server, 6 sessions have to be considered to cover all possible scenarios with two honest agents A, B and a dishonest one C (A talking to B, A talking to C, and C talking to B, and three additional sessions where the role of the agents A and B are swapped). Assuming that dishonest roles do not need to be modelled, this leads to a scenario with 14 roles in parallel. In practice, an attack does not require 14 sessions. 3-4 sessions are typically sufficient. However, it is impossible to predict which scenario is required for the attack. Moreover, since the problem of deciding equivalence is actually undecidable, an attack may require an arbitrary number of sessions. Therefore, the more sessions we can check, the more confidence we obtain.

Our contribution. In this paper, we propose a different procedure for deciding equivalence. Instead of designing a crafted algorithm for equivalence, we use more general verification techniques, namely Graph planning [14], [15] and SAT-solving. The idea of using Graph planning and SAT-solvers for analyzing protocols has already been explored in [16], yielding the tool SATMC [17] for trace properties. Moving from trace to equivalence properties is far from being straightforward as exemplified by the research effort on equivalence these past 10 years (see e.g. [18] for a survey).

Let us first sketch how SATMC works. The tool focuses at secrecy and encodes accessibility of a (secret) term into a SAT formula. For efficiency reasons, the main step of SATMC actually consists in applying Graph planning techniques in order to compute an over-approximation of reachable messages. If no secret has been found, the protocol is deemed secure. Otherwise, actual accessibility of the potentially leaked secret is encoded into a SAT formula.

In order to benefit from Graph planning and SAT-solvers, the size of messages has to be bounded and this bound needs to be practical. In [16], [17], the authors simply assume protocols to be given with a (finite) format for the messages. Here, we do not bound *a priori* the format of the messages. Instead, we rely on a recent result [19] that shows that if there is

an attack, that is a witness of non equivalence between two protocols P and Q, then there is a "small" attack, where messages comply to a certain format (induced by a type). This result holds for deterministic protocols that use symmetric keys and pairing. Note that this result only controls the format of the messages exchanged in P, not in Q (or conversely). The fact that the messages in Q are *a priori* unbounded forbids any direct encoding of (non) equivalence into a SAT formula. Planning graphs are particularly helpful here: while computing an over-approximation of the messages reachable in P, we simultaneously obtained an over-approximation of the messages that need to be considered in Q for checking equivalence w.r.t. P. This requires of course to characterize (non) equivalence as a reachability property, which is made possible thanks to the protocols' determinism.

In order to further reduce the traces than need to be explored, we show that we can restrict ourselves to an attacker that only *decompose* messages (and do not compose them), provided that protocols are *flattened*, that is all meaningful composition steps are pre-computed in advance. This flattening technique has been used in [16] (although we are not aware of any proof of correction). We formally prove this technique to be sound, in the more general case of equivalence properties. Handling equivalence is non trivial since it is not sufficient to preserve the set of messages that can be computed, it is also necessary to preserve cases of failure on both processes. Moreover, we had use one more ingredient to obtain an efficient bound. We significantly reduce the number of constants that need to be considered to find an attack. Namely, we show that only two constants are necessary, which is a result of independent interest.

Implementation. We have implemented our algorithm and our first experimentations demonstrate the good performance of our tool. For most protocols, we can easily analyse several sessions while the three other tools (Akiss, Spec, Apte) typically fail for more than one session, with the exception of the variant Apte-por [13], which can handle several sessions, in some cases. All files related to the tool implementation and case studies are available at [20], while omitted proofs are available in the full version of this paper [21].

II. Model for security protocols

A common framework for modelling security protocols are process algebra like the applied pi-calculus [22]. We consider here a variant of the calculus provided in [19] in order to benefit from its main result, which guarantees a "small attack" property: when there is an attack, there is a well-typed attack.

A. Syntax

Term algebra: As usual, messages are modelled by terms. We consider an infinite set of *names* $\mathcal{N}$, an infinite set of constants Σ_0, and two distinct sets of *variables* $\mathcal{X}$ and $\mathcal{W}$. Names are typically used to represent keys or nonces. Variables in $\mathcal{X}$ refer to unknown parts of messages expected by participants while variables in $\mathcal{W}$ are used to store messages learnt by the attacker. We consider the following sets of function symbols:

$$\Sigma_{\mathsf{c}} = \{\mathsf{enc}, \langle\ \rangle\} \quad \Sigma_{\mathsf{d}} = \{\mathsf{dec}, \mathsf{proj}_1, \mathsf{proj}_2\} \quad \Sigma_{\mathsf{std}} = \Sigma_{\mathsf{c}} \cup \Sigma_{\mathsf{d}}$$

The symbol enc and dec both of arity 2 represent encryption and decryption. Concatenation of messages is modelled through the symbol $\langle\ \rangle$ of arity 2, with projection functions proj_1 and proj_2 of arity 1. We distinguish between *constructor* symbols in Σ_{c} and *destructor* symbols in Σ_{d}.

We consider several sets of terms. Given a set of A of atoms (*i.e.* names, variables, and constants), and a signature $\mathcal{F} \in \{\Sigma_{\mathsf{c}}, \Sigma_{\mathsf{d}}, \Sigma_{\mathsf{std}}\}$, we denote by $\mathcal{T}(\mathcal{F}, \mathsf{A})$ the set of terms built from $\mathcal{F}$ and A. Constructors terms with atomic encryptions are represented by the set $\mathcal{T}_0(\Sigma_{\mathsf{c}}, \mathsf{A})$, which is the subset of $\mathcal{T}(\Sigma_{\mathsf{c}}, \mathsf{A})$ such that any subterm $\mathsf{enc}(m, k)$ of a term in $\mathcal{T}_0(\Sigma_{\mathsf{c}}, \mathsf{A})$ is such that $k \in \mathsf{A}$. Given $\Sigma \subseteq \Sigma_0$, we denote by $\mathcal{M}_\Sigma$ the set $\mathcal{T}_0(\Sigma_{\mathsf{c}}, \Sigma \cup \mathcal{N})$, *i.e.* the set of *messages* built using constants in Σ. The *positions* of a term are defined as usual. We denote $vars(u)$ the set of variables that occur in u. The application of a substitution σ to a term u is written $u\sigma$, and we denote $dom(\sigma)$ its *domain*, and $img(\sigma)$ its *image*. Two terms u_1 and u_2 are *unifiable* when there exists σ such that $u_1\sigma = u_2\sigma$. In this case, we denote mgu their most general unifier. The composition of two substitutions σ_1 and σ_2 is denoted $\sigma_1 \circ \sigma_2$.

Example 1: Let k_{ab} and k_{bs} be two names in $\mathcal{N}$, and a be a constant from Σ_0. We have that $t = \mathsf{enc}(\langle k_{ab}, \mathsf{a}\rangle, k_{as})$ is a message from $\mathcal{M}_{\Sigma_0}$, whereas $\mathsf{enc}(\mathsf{a}, \langle k_{as}, k_{as}\rangle)$ is not (due to the presence of a compound term in key position).

An attacker can build any term by applying function symbols. His computation is formally modelled by terms, called *recipes*. Given $\Sigma \subseteq \Sigma_0$, we denote $\mathcal{R}_\Sigma$ the set $\mathcal{T}(\Sigma_{\mathsf{std}}, \Sigma \cup \mathcal{W})$, *i.e.* the set of recipes built using constants in Σ. Note that a recipe does not contain names, since, intuitively, names are initially secret.

Example 2: Assume that the attacker has first intercepted the message t (stored in w_1), and then the key k_{as} (stored in w_2). The term $R = \mathsf{proj}_1(\mathsf{dec}(\mathsf{w}_1, \mathsf{w}_2))$ is a recipe that represents a computation that can be performed by the attacker. Indeed, he can decrypt the first message with the second one, and then apply a projection operator.

The decryption of an encryption with the right key yields the plaintext. Similarly, the left (or right) projection of a concatenation yields the left (or right) component. These properties are reflected in the three following convergent rewrite rules:

$$\mathsf{dec}(\mathsf{enc}(x, y), y) \to x, \text{and } \mathsf{proj}_i(\langle x_1, x_2\rangle) \to x_i \ \ i \in \{1, 2\}.$$

A term u can be rewritten in v if there is a position p in u, and a rewriting rule $\mathsf{g}(t_1, \ldots, t_n) \to t$ such that $u|_p = \mathsf{g}(t_1, \ldots, t_n)\theta$ for some substitution θ. Moreover, we assume that $t_1\theta, \ldots, t_n\theta$ as well as $t\theta$ are *messages*. This assumption slightly differs from [19]. Here, whenever an inner decryption/projection fails then the overall evaluation fails. Intuitively, we model eager evaluation while [19] models lazy evaluation. Our rewriting system is convergent, and we denote $u{\downarrow}$ the *normal form* of a given term u.

Example 3: Let t be the term given in Example 1, we have that $\mathsf{proj}_1(\mathsf{dec}(t, k_{as}))\downarrow = k_{ab}$. Indeed, we have that:

$$\begin{aligned}\mathsf{proj}_1(\mathsf{dec}(t, k_{as})) &= \mathsf{proj}_1(\mathsf{dec}(\mathsf{enc}(\langle k_{ab}, \mathsf{a}\rangle, k_{as}), k_{as})) \\ &\to \mathsf{proj}_1(\langle k_{ab}, \mathsf{a}\rangle) \\ &\to k_{ab}\end{aligned}$$

Process algebra: We only consider public channels and we assume that each process communicates on a dedicated channel. In practice, an attacker can typically distinguish between protocol participants thanks to their IP address and even between protocol sessions thanks to session identifiers. Technically, this assumption avoids non determinism. Formally, we assume an infinite set $\mathcal{C}h$ of channels and we consider the fragment of simple processes without replication built on basic processes as defined e.g. in [23]. A basic process represents a party in a protocol, which may sequentially perform actions such as waiting for a message of a certain form, and outputting a message. Then, a simple process is a parallel composition of such basic processes playing on distinct channels.

Definition 1: The set of *basic processes* on $c \in \mathcal{C}h$ is defined as follows (with $u_1, u_2 \in \mathcal{T}(\Sigma_{\mathsf{c}}, \Sigma_0 \cup \mathcal{N} \cup \mathcal{X})$):

$$P, Q := 0 \mid \mathsf{in}(c, u_1).P \mid \mathsf{out}(c, u_2).P$$

A *simple process* $\mathcal{P} = \{P_1, \ldots, P_n\}$ is a multiset of basic processes P_i on pairwise distinct channels c_i.

The process 0 does nothing. The process "$\mathsf{in}(c, u_1).P$" expects a message m of the form u_1 on channel c and then behaves like $P\sigma$ where σ is a substitution such that $m = u_1\sigma$ is a message. The process "$\mathsf{out}(c, u_2).P$" emits u_2 on channel c, and then behaves like P. We assume that names are implicitly freshly generated, and therefore we do need a specific action to model name generation. The construction "new" is important in the presence of replication but we do not consider replication here. For the sake of clarity, we may omit the null process. We write $fv(P)$ for the set of *free variables* that occur in P, *i.e.* the set of variables that are not in the scope of an input.

Definition 2: A *protocol* is a simple process $\mathcal{P}$ that is ground, *i.e.* $fv(\mathcal{P}) = \emptyset$.

Example 4: The Denning Sacco protocol [24] (without timestamps) is a key distribution protocol using symmetric encryption and a trusted server. Informally, we have:

1. $A \to S: \ A, B$
2. $S \to A: \ \{B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$
3. $A \to B: \ \{K_{ab}, A\}_{K_{bs}}$

where $\{m\}_k$ denotes the symmetric encryption of a message m with key k. The agents A and B aim at authenticating each other and establishing a session key K_{ab} through a trusted server S. The key K_{as} (resp. K_{bs}) is a long term key shared between A and S (resp. B and S).

To model the Denning Sacco protocol, we introduce several atomic data: k_{as}, k_{bs}, k_{ab} are names, a and b are constants from Σ_0, and c_1, c_2, and c_3 are channel names from $\mathcal{C}h$. Each role is modelled by a basic process that is described below. Below, we denote by $\langle x_1, x_2, x_3\rangle$ the term $\langle x_1, \langle x_2, x_3\rangle\rangle$.

$$\begin{aligned}P_A &= \mathsf{out}(c_1, \langle \mathsf{a}, \mathsf{b}\rangle). \\ &\quad \mathsf{in}(c_1, \mathsf{enc}(\langle \mathsf{b}, x_{AB}, x_B\rangle, k_{as})). \\ &\quad \mathsf{out}(c_1, x_B) \\ P_S &= \mathsf{in}(c_2, \langle \mathsf{a}, \mathsf{b}\rangle). \\ &\quad \mathsf{out}(c_2, \mathsf{enc}(\langle \mathsf{b}, k_{ab}, \mathsf{enc}(\langle k_{ab}, \mathsf{a}\rangle, k_{bs})\rangle, k_{as})) \\ P_B &= \mathsf{in}(c_3, \mathsf{enc}(\langle y_{AB}, \mathsf{a}\rangle, k_{bs}))\end{aligned}$$

The protocol is then modelled by the simple ground process $\mathcal{P}_{\mathsf{DS}} = \{P_A, P_S, P_B\}$. In order to model several sessions of the same protocol, we simply have to consider several instances of the basic processes P_A, P_S, and P_B. We will use different channel names to get a simple process, different names to model fresh names, and we will rename variables to avoid clashes. Two sessions of the Denning-Sacco protocol (between honest participants) are therefore modelled by:

$$\mathcal{P}'_{\mathsf{DS}} = \{P_A, P_S, P_B, P'_A, P'_S, P'_B\}$$

where P'_A, P'_B, and P'_S are given below:

$$\begin{aligned}P'_A &= \mathsf{out}(c_4, \langle \mathsf{a}, \mathsf{b}\rangle). \\ &\quad \mathsf{in}(c_4, \mathsf{enc}(\langle \mathsf{b}, x'_{AB}, x'_B\rangle, k_{as})). \\ &\quad \mathsf{out}(c_4, x'_B) \\ P'_S &= \mathsf{in}(c_5, \langle \mathsf{a}, \mathsf{b}\rangle). \\ &\quad \mathsf{out}(c_5, \mathsf{enc}(\langle \mathsf{b}, k'_{ab}, \mathsf{enc}(\langle k'_{ab}, \mathsf{a}\rangle, k_{bs})\rangle, k_{as})) \\ P'_B &= \mathsf{in}(c_6, \mathsf{enc}(\langle y'_{AB}, \mathsf{a}\rangle, k_{bs}))\end{aligned}$$

B. Semantics

The operational semantics of a process is defined using a relation over configurations, *i.e.* triples $(\mathcal{P}; \phi; \sigma)$ where:

- $\mathcal{P}$ is a multiset of processes with $fv(\mathcal{P}) \subseteq dom(\sigma)$;
- ϕ is a *frame*, *i.e.* a substitution of the form
$$\{\mathsf{w}_1 \triangleright m_1, \ldots, \mathsf{w}_n \triangleright m_n\}$$
where $\mathsf{w}_1, \ldots, \mathsf{w}_n \in \mathcal{W}$, and $m_1, \ldots, m_n \in \mathcal{M}_{\Sigma_0}$;
- σ is a substitution such that $dom(\sigma) \subseteq \mathcal{X}$, and $img(\sigma) \subseteq \mathcal{M}_{\Sigma_0}$.

We often write $\mathcal{P}$ instead of $(\mathcal{P}; \emptyset; \emptyset)$, and $P \cup \mathcal{P}$ instead of $\{P\} \cup \mathcal{P}$. The terms in ϕ represent the messages that are sent out and therefore known by the attacker whereas the substitution σ is used to store parts of the messages received so far. The operational semantics of a process is induced by the relation $\xrightarrow{\alpha}$ over configurations defined below:

IN
$(\mathsf{in}(c, u).P \cup \mathcal{P};\ \phi;\ \sigma) \xrightarrow{\mathsf{in}(c,R)} (P \cup \mathcal{P};\ \phi;\ \sigma \uplus \sigma_0)$
where $R \in \mathcal{R}_{\Sigma_0}$ such that $R\phi\downarrow \in \mathcal{M}_{\Sigma_0}$,
and $R\phi\downarrow = (u\sigma)\sigma_0$ for σ with $dom(\sigma_0) = vars(u\sigma)$.

OUT
$(\mathsf{out}(c, u).P \cup \mathcal{P};\ \phi;\ \sigma) \xrightarrow{\mathsf{out}(c,\mathsf{w})} (P \cup \mathcal{P};\ \phi \cup \{\mathsf{w} \triangleright u\sigma\};\ \sigma)$
with w a fresh variable from $\mathcal{W}$, and $u\sigma \in \mathcal{M}_{\Sigma_0}$.

A process may input any term that an attacker can built from publicly available terms and symbols (rule IN). The second rule corresponds to the output of a term: the corresponding term is added to the frame of the current configuration, which

means that the attacker has now access to it. Note that the term is outputted provided that it is a message. In case the evaluation of the term yields an encryption with a non atomic key, the evaluation fails and there is no output. We do not need to model internal communications since we assume public channels: all communications are controlled by the attacker.

The relation $\xrightarrow{\mathsf{tr}}$ between configurations (where tr is a possibly empty sequence of actions) is defined in the usual way. Given $\Sigma \subseteq \Sigma_0$, and a protocol $\mathcal{P}$ we define its *set of traces w.r.t.* Σ as follows:

$$\mathsf{trace}_\Sigma(\mathcal{P}) = \{(\mathsf{tr}, \phi) \mid (\mathcal{P}; \emptyset; \emptyset) \xrightarrow{\mathsf{tr}} (\mathcal{P}'; \phi; \sigma) \text{ for some configuration } (\mathcal{P}'; \phi; \sigma) \text{ and any recipe occurring in } \mathsf{tr} \text{ is in } \mathcal{R}_\Sigma.\}.$$

Note that, for any $(\mathsf{tr}, \phi) \in \mathsf{trace}_\Sigma(\mathcal{P})$, we have that $\mathsf{tr}\phi\downarrow$ only contains messages in $\mathcal{M}_{\Sigma_0}$.

Example 5: Consider the following sequence tr:

$$\mathsf{tr} = \mathsf{out}(c_1, \mathsf{w}_1).\mathsf{in}(c_2, \mathsf{w}_1).\mathsf{out}(c_2, \mathsf{w}_2).\mathsf{in}(c_1, \mathsf{w}_2).\mathsf{out}(c_1, \mathsf{w}_3).\mathsf{in}(c_3, \mathsf{w}_3)$$

This sequence tr allows one to reach the frame:

$$\phi = \{\mathsf{w}_1 \triangleright \langle \mathsf{a}, \mathsf{b}\rangle,\ \mathsf{w}_2 \triangleright \mathsf{enc}(\langle \mathsf{b}, k_{ab}, \mathsf{enc}(\langle k_{ab}, \mathsf{a}\rangle, k_{bs})\rangle, k_{as}),\ \mathsf{w}_3 \triangleright \mathsf{enc}(\langle k_{ab}, \mathsf{a}\rangle, k_{bs})\ \}.$$

We have that $(\mathsf{tr}, \phi) \in \mathsf{trace}_\Sigma(\mathcal{P}_{\mathsf{DS}})$. This trace corresponds to a normal execution of the protocol.

C. Trace equivalence

Trace equivalence can be used to formalise many interesting security properties, in particular privacy-type properties. We assume keys to be atomic and encryption to fail for non atomic keys. We define trace equivalence accordingly, by letting the attacker observe when an encryption fails. We first define equivalence on sequences of messages.

Definition 3: A frame ϕ_1 is *statically included* w.r.t. $\Sigma \subseteq \Sigma_0$ in a frame ϕ_2, denoted $\phi_1 \sqsubseteq_s \phi_2$, when we have that $dom(\phi_1) = dom(\phi_2)$, and:

- for any $R \in \mathcal{R}_\Sigma$, $R\phi_1\downarrow \in \mathcal{M}_{\Sigma_0}$ implies that $R\phi_2\downarrow \in \mathcal{M}_{\Sigma_0}$;
- for any $R_1, R_2 \in \mathcal{R}_\Sigma$ with $R_1\phi_1\downarrow, R_2\phi_1\downarrow \in \mathcal{M}_{\Sigma_0}$, $R_1\phi_1\downarrow = R_2\phi_1\downarrow$ implies that $R_1\phi_2\downarrow = R_2\phi_2\downarrow$.

They are in *static equivalence* w.r.t. Σ, denoted $\phi_1 \sim_s \phi_2$, when $\phi_1 \sqsubseteq_s \phi_2$, and $\phi_2 \sqsubseteq_s \phi_1$ (both w.r.t. Σ).

Example 6: Consider $\phi_1 = \phi \cup \{\mathsf{w}_4 \triangleright \mathsf{enc}(\mathsf{m}_1, k_{ab})\}$ and $\phi_2 = \phi \cup \{\mathsf{w}_4 \triangleright \mathsf{enc}(\mathsf{m}_2, k)\}$ where ϕ has been introduced in Example 5. The terms m_1, m_2 are public constants from Σ_0, and k is a name from $\mathcal{N}$. We have that the two frames ϕ_1 and ϕ_2 are statically equivalent (w.r.t. any Σ). Intuitively, at the end of a normal execution between honest participants, an attacker can not distinguish whether the key used to encrypt a message (here the constants m_1 and m_2) is the session key that has been established or a fresh key k.

In contrast, the frames $\phi_1' = \phi_1 \cup \{\mathsf{w}_5 \triangleright k_{ab}\}$ and $\phi_2' = \phi_2 \cup \{\mathsf{w}_5 \triangleright k_{ab}\}$ are *not* in static equivalence. Actually ϕ_1' is not statically included in ϕ_2'. Indeed, an attacker can observe that the 4th message of ϕ_1 can be decrypted by the 5th message, which is not the case in ϕ_2'. Formally, considering $R = \mathsf{dec}(\mathsf{w}_4, \mathsf{w}_5)$, we have $R\phi_1'\downarrow \in \mathcal{M}_{\Sigma_0}$ while $R\phi_2'\downarrow \notin \mathcal{M}_{\Sigma_0}$.

Then, we lift this notion of equivalence from frames to configurations.

Definition 4: Let $\Sigma \subseteq \Sigma_0$. A protocol $\mathcal{P}$ is *trace included* w.r.t. Σ in a protocol $\mathcal{Q}$, written $\mathcal{P} \sqsubseteq_t \mathcal{Q}$, if for every $(\mathsf{tr}, \phi) \in \mathsf{trace}_\Sigma(\mathcal{P})$, there exists $(\mathsf{tr}', \psi) \in \mathsf{trace}_\Sigma(\mathcal{Q})$ such that $\mathsf{tr} = \mathsf{tr}'$ and $\phi \sqsubseteq_s \psi$ w.r.t. Σ. The protocols $\mathcal{P}$ and $\mathcal{Q}$ are *trace equivalent* w.r.t. Σ, written $\mathcal{P} \approx_t \mathcal{Q}$, if $\mathcal{P} \sqsubseteq_t \mathcal{Q}$ and $\mathcal{Q} \sqsubseteq_t \mathcal{P}$ (both w.r.t. Σ).

This notion of equivalence (even when $\Sigma = \Sigma_0$) does not coincide in general with the usual notion of trace equivalence as defined e.g. in [23]. It is actually coarser since we simply require the resulting frames to be in static inclusion ($\phi \sqsubseteq_s \psi$) instead of static equivalence ($\phi \sim_s \psi$). However, these two notions actually coincide (see [10]) for the class of simple processes that we consider in this paper.

Assume given two protocols $\mathcal{P}$ and $\mathcal{Q}$ such that $\mathcal{P} \not\sqsubseteq_t \mathcal{Q}$ w.r.t. Σ. A witness of this non-inclusion is a trace tr w.r.t. Σ for which there exists ϕ such that $(\mathsf{tr}, \phi) \in \mathsf{trace}_\Sigma(\mathcal{P})$ and:

- either there is no ψ such that $(\mathsf{tr}, \psi) \in \mathsf{trace}_\Sigma(\mathcal{Q})$;
- or such ψ exists and $\phi \not\sqsubseteq_s \psi$ w.r.t. Σ.

Note that for a simple process, once the sequence tr is fixed, the resulting frame reachable through tr is uniquely defined (when it exists) since simple processes are deterministic.

Example 7: The protocol $\mathcal{P}'_{\mathsf{DS}}$ presented in Example 4 models two sessions of the Denning Sacco protocol. Assume now that we wish to check strong secrecy of the exchanged key, as received by the agent A. This can be expressed by checking whether $\mathcal{P}'^1_{\mathsf{DS}} \approx_t \mathcal{P}'^2_{\mathsf{DS}}$ where:

- $\mathcal{P}'^1_{\mathsf{DS}}$ is as $\mathcal{P}'_{\mathsf{DS}}$ but we add "$\mathsf{out}(c_1, \mathsf{enc}(\mathsf{m}_1, x_{AB}))$" at the end of the process P_A, and "$\mathsf{out}(c_4, \mathsf{enc}(\mathsf{m}_1, x'_{AB}))$" at the end of P'_A
- $\mathcal{P}^2_{\mathsf{DS}}$ is as $\mathcal{P}_{\mathsf{DS}}$ but we add the instruction "$\mathsf{out}(c_1, \mathsf{enc}(\mathsf{m}_2, k))$" at the end of P_A, and "$\mathsf{out}(c_4, \mathsf{enc}(\mathsf{m}_2, k'))$" at the end of P'_A.

The terms m_1 and m_2 are two public constants from Σ_0 whereas k and k' are names from $\mathcal{N}$.

While the key received by A cannot be learnt by an attacker, strong secrecy of this key is not guaranteed. Indeed, due to the lack of freshness, the same key can be sent several times to A, and this can be observed by an attacker. Formally, the attack is as follows. Consider the sequence

$$\mathsf{tr}' = \mathsf{tr} \cdot \mathsf{out}(c_4, \mathsf{w}_4).\mathsf{in}(c_4, \mathsf{w}_2).\mathsf{out}(c_4, \mathsf{w}_5).\mathsf{out}(c_1, \mathsf{w}_6).\mathsf{out}(c_4, \mathsf{w}_7)$$

where tr has been defined in Example 5. The attacker simply replays an old session. The resulting (unique) frames are

- $\phi_1' = \phi \cup \phi' \cup \{\mathsf{w}_6 \triangleright \mathsf{enc}(\mathsf{m}_1, k_{ab}),\ \mathsf{w}_7 \triangleright \mathsf{enc}(\mathsf{m}_1, k_{ab})\}$,
- $\phi_2' = \phi \cup \phi' \cup \{\mathsf{w}_6 \triangleright \mathsf{enc}(\mathsf{m}_2, k),\ \mathsf{w}_7 \triangleright \mathsf{enc}(\mathsf{m}_2, k')\}$

where ϕ is the frame given in Example 5, and

$$\phi' = \{\mathsf{w}_4 \triangleright \langle \mathsf{a}, \mathsf{b}\rangle, \mathsf{w}_5 \triangleright \mathsf{enc}(\langle k_{ab}, \mathsf{a}\rangle, k_{bs})\}.$$

We have that $(\mathsf{tr}', \phi_1') \in \mathsf{trace}_{\Sigma_0}(\mathcal{P}'^1_{\mathsf{DS}})$ and $(\mathsf{tr}', \phi_2') \in \mathsf{trace}_{\Sigma_0}(\mathcal{P}'^2_{\mathsf{DS}})$. However, we have that $\phi_1' \not\sqsubseteq_s \phi_2'$ since $\mathsf{w}_6 = \mathsf{w}_7$ in ϕ_1' but not in ϕ_2'. Thus $\mathcal{P}'^1_{\mathsf{DS}}$ is *not* trace included in $\mathcal{P}'^2_{\mathsf{DS}}$. To avoid this attack, the messages of the Denning-Sacco protocol shall include timestamps or nonces.

The goal of the paper is to provide an efficient and practical procedure for checking trace equivalence.

III. Reduction results

Even when considering finite processes (*i.e.* processes without replication), the problem of checking trace equivalence is difficult due to several sources of unboundedness:

- the size of messages which can be forged by an attacker is unbounded;
- the number of nonces and constants that can be used by an attacker is unbounded too.

Recently, [19] has established how to reduced the search space for attacks by bounding the size of messages involved in a minimal attack. From a theoretical point of view, this also yields a bound on the number of nonces/constants involved in such a minimal attack. However, this bound is far from being practical. In this section, we show that the small attack property of [19] still holds even if our semantics has slightly changed (due to eager evaluation) and we further demonstrate that the number of constants can be significantly reduced since only three constants need to be considered (and no nonces), in addition to those explicitly mentioned in the protocol.

A. Bounding the size of messages

As in [19], we consider type-compliant protocols, and we restrict ourselves to typing systems that preserve the structure of terms. A typing system is defined as follows.

Definition 5: A *typing system* is a pair $(\mathcal{T}_0, \delta_0)$ where $\mathcal{T}_0$ is a set of elements called *atomic types* with a special atomic type denoted $\tau_\star$, and δ_0 is a function mapping atomic terms in $\Sigma_0 \cup \mathcal{N} \cup \mathcal{X}$ to types τ generated using the following grammar:

$$\tau, \tau_1, \tau_2 = \tau_0 \mid \langle \tau_1, \tau_2 \rangle \mid \mathsf{enc}(\tau_1, \tau_2) \text{ with } \tau_0 \in \mathcal{T}_0.$$

We further assume the existence of an infinite number of constants in Σ_0 (resp. variables in $\mathcal{X}$, names in $\mathcal{N}$) of any type, and the existence of three special constants denoted $\mathsf{c}_{\langle\omega,\omega\rangle}$, $\mathsf{c}^0_\star$, and $\mathsf{c}^1_\star$ of type $\tau_\star$. The constant $\mathsf{c}_{\langle\omega,\omega\rangle}$ can not be used in key position. Then, δ_0 is extended to constructor terms as follows:

$$\delta_0(\mathsf{f}(t_1, \ldots, t_n)) = \mathsf{f}(\delta_0(t_1), \ldots, \delta_0(t_n)) \text{ with } \mathsf{f} \in \Sigma_{\mathsf{c}}.$$

Example 8: Continuing our running Example, we consider the typing system generated from the set $\mathcal{T}_{\mathsf{DS}} = \{\tau_{\mathsf{a}}, \tau_{\mathsf{m}}, \tau_{ks}, \tau_k\}$ of atomic types and the function δ_{DS} that associates the expected type to each constant/name, and the following types to variables:

- $\delta_{\mathsf{DS}}(x_{AB}) = \delta_{\mathsf{DS}}(x'_{AB}) = \delta_{\mathsf{DS}}(y_{AB}) = \delta_{\mathsf{DS}}(y'_{AB}) = \tau_k$;
- $\delta_{\mathsf{DS}}(x_B) = \delta_{\mathsf{DS}}(x'_B) = \mathsf{enc}(\langle \tau_k, \tau_{\mathsf{a}} \rangle, \tau_{ks})$.

A protocol is type-compliant if two unifiable subterms have the same type. Formally, we use the definition given in [19], which is similar to the one introduced in [25].

We write $St(t)$ (resp. $St(\tau)$) for the set of *(syntactic) subterms* of a term t (resp. type τ), and $ESt(t)$ the set of its *encrypted subterms*, *i.e.*

$$ESt(t) = \{u \in St(t) \mid u \text{ is of the form } \mathsf{enc}(u_1, u_2)\}.$$

In the following definition, $\delta_{\mathcal{P}}(P)$ is the set of $\delta_{\mathcal{P}}(t)$ for every term t occuring in protocol P.

Definition 6: A protocol $\mathcal{P}$ is *type-compliant* w.r.t. a typing system $(\mathcal{T}_{\mathcal{P}}, \delta_{\mathcal{P}})$ if $\tau_\star \notin St(\delta_{\mathcal{P}}(\mathcal{P}))$, and for every $t, t' \in ESt(\mathcal{P})$ we have that:

$$t \text{ and } t' \text{ unifiable implies that } \delta_{\mathcal{P}}(t) = \delta_{\mathcal{P}}(t').$$

Example 9: The protocol $\mathcal{P}'^1_{\mathsf{DS}}$ (resp. $\mathcal{P}'^2_{\mathsf{DS}}$) is type-compliant w.r.t. the typing system given in Example 8. Indeed, the encrypted subterms of $\mathcal{P}'^1_{\mathsf{DS}}$ are:

1) $t_A = \mathsf{enc}(\langle \mathsf{b}, x_{AB}, x_B \rangle, k_{as})$;
2) $t_{B1} = \mathsf{enc}(\langle y_{AB}, \mathsf{a} \rangle, k_{bs})$;
3) $t_{B2} = \mathsf{enc}(\mathsf{m}_1, y_{AB})$;
4) $t_{S1} = \mathsf{enc}(\langle \mathsf{b}, k_{ab}, \mathsf{enc}(\langle k_{ab}, \mathsf{a} \rangle, k_{bs}) \rangle, k_{as})$;
5) $t_{S2} = \mathsf{enc}(\langle k_{ab}, \mathsf{a} \rangle, k_{bs})$

as well as the renaming of these terms obtained by replacing k_{ab}, x_{AB}, y_{AB}, and x_B with fresh names/variables of the same type, namely k'_{ab}, x'_{AB}, y'_{AB}, and x'_B.

It is easy to check that the type-compliance condition is satisfied for any pair of terms. For instance, we have that t_A and t_{S1} are unifiable, and they have indeed the same type:

$$\delta_{\mathsf{DS}}(t_A) = \mathsf{enc}(\langle \tau_{\mathsf{a}}, \tau_k, \mathsf{enc}(\langle \tau_k, \tau_{\mathsf{a}} \rangle, \tau_{ks}) \rangle, \tau_{ks}) = \delta_{\mathsf{DS}}(t_{S1}).$$

Consider a protocol $\mathcal{P}$ that is type-compliant w.r.t. to a typing system $(\mathcal{T}_{\mathcal{P}}, \delta_{\mathcal{P}})$, an execution $\mathcal{P} \xrightarrow{\mathsf{tr}} (\mathcal{P}'; \phi'; \sigma')$ is *well-typed* if σ' is a well-typed substitution, *i.e.* every variable of its domain has the same type as its image. We say that a trace $(\mathsf{tr}, \phi) \in \mathsf{trace}_{\Sigma}(\mathcal{P})$ is well-typed if its underlying execution (unique due to the class of protocols we consider in this paper) is well-typed. Given a protocol $\mathcal{P}$, we denote $\Sigma_{\mathcal{P}}$ the constants from Σ_0 that occur in $\mathcal{P}$.

We first show that the small attack property from [19] still holds: whenever two processes are not in trace equivalence, then there is a well-typed witness of non equivalence. In addition, we show that the recipes involved in such a trace have a simple form: they are built using constructor symbols on top of destructors.

Definition 7: Let R be a recipe. We say that R is *destructor-only* if $R \in \mathcal{T}(\Sigma_{\mathsf{d}}, \Sigma \cup \mathcal{W})$. It is *simple* if there exist destructor-only recipes $R_1, \ldots, R_k$, and a context C made of constructors such that $R = C[R_1, \ldots, R_k]$.

Theorem 1: Let $\mathcal{P}$ be a protocol type-compliant w.r.t. $(\mathcal{T}_{\mathcal{P}}, \delta_{\mathcal{P}})$ and $\mathcal{Q}$ be another protocol. We have that $\mathcal{P} \not\sqsubseteq_t \mathcal{Q}$ w.r.t. Σ_0 if, and only if, there exists a witness tr of this non-inclusion that only contains simple recipes and such that one of the following holds:

1) $(\mathsf{tr}, \phi) \in \mathsf{trace}_{\Sigma_0}(\mathcal{P})$ for some ϕ and (tr, ϕ) is well-typed w.r.t. $(\mathcal{T}_{\mathcal{P}}, \delta_{\mathcal{P}})$;

2) $\mathsf{tr} = \mathsf{tr}'\{\mathsf{c}_0 \mapsto \mathsf{c}_{\langle\omega,\omega\rangle}\}$ for some $\mathsf{c}_0 \in \Sigma_0 \setminus \Sigma_{\mathcal{P}}$, and $(\mathsf{tr}', \phi') \in \mathsf{trace}_{\Sigma_0}(\mathcal{P})$ for some ϕ' and (tr', ϕ') is well-typed w.r.t. $(\mathcal{T}_{\mathcal{P}}, \delta_{\mathcal{P}})$.

Since we consider atomic keys, some execution may fail when a protocol is about to output an encryption with a non atomic key. In order to detect this kind of behaviours, it is important to consider slightly ill-typed traces as defined in Item 2.

Example 10: Continuing our running example, we have seen that $\mathcal{P}'^1_{\mathsf{DS}} \not\sqsubseteq_t \mathcal{P}'^2_{\mathsf{DS}}$. The witness tr' of this non-inclusion (given in Example 7) only contains simple recipes, and $(\mathsf{tr}', \phi'_1) \in \mathsf{trace}_{\Sigma_0}(\mathcal{P}'^1_{\mathsf{DS}})$ is well-typed w.r.t. $(\mathcal{T}_{\mathsf{DS}}, \delta_{\mathsf{DS}})$ (the typing system given in Example 8).

B. Bounding the number of constants

The previous result implicitly bounds the number of constants used in an attack but the induced bound would be impractical. We show here that actually, two constants are sufficient. The proof technique is inspired from [26] and [27] which respectively reduce the number of nonces and agents in the context of equivalence properties. A direct application of the proof technique would however yield two constants of each type, which represents still a high number of constants. Instead, we show here that just two constants are enough, provided they are of special type $\tau_\star$. To obtain this result, we slightly relax the notion of well-typedness.

Given a typing system $(\mathcal{T}_0, \delta_0)$, we denote by $\preceq$ the smallest relation on types defined as follows:

- $\tau_\star \preceq \tau$ and $\tau \preceq \tau$ for any type τ (atomic or not);
- $\mathsf{f}(\tau_1, \tau_2) \preceq \mathsf{f}(\tau'_1, \tau'_2)$ when $\tau_1 \preceq \tau'_1$, $\tau_2 \preceq \tau'_2$, and $\mathsf{f} \in \Sigma_c$.

Consider a protocol $\mathcal{P}$ that is type-compliant w.r.t. to a typing system $(\mathcal{T}_{\mathcal{P}}, \delta_{\mathcal{P}})$, an execution $\mathcal{P} \xrightarrow{\mathsf{tr}} (\mathcal{P}'; \phi'; \sigma')$ is *quasi-well-typed* if $\delta_{\mathcal{P}}(x\sigma') \preceq \delta_{\mathcal{P}}(x)$ for every variable $x \in dom(\sigma')$. We say that a trace $(\mathsf{tr}, \phi) \in \mathsf{trace}_\Sigma(\mathcal{P})$ is quasi-well-typed if its underlying execution (unique due to the class of protocols we consider in this paper) is quasi-well-typed.

If two processes are not in trace equivalence, then there is a witness of non equivalence that is quasi-well typed and uses at most two extra constants plus eventually $\mathsf{c}_{\langle\omega,\omega\rangle}$ to detect slightly ill-typed traces.

Theorem 2: Let $\mathcal{P}$ be a protocol type-compliant w.r.t. $(\mathcal{T}_{\mathcal{P}}, \delta_{\mathcal{P}})$ and $\mathcal{Q}$ be another protocol. Let $\Sigma = \Sigma_{\mathcal{P}} \uplus \{\mathsf{c}^0_\star, \mathsf{c}^1_\star, \mathsf{c}_{\langle\omega,\omega\rangle}\}$. We have that $\mathcal{P} \not\sqsubseteq_t \mathcal{Q}$ w.r.t. Σ_0 if, and only if, there exists a witness tr of this non-inclusion w.r.t. Σ that only contains simple recipes, and such that $(\mathsf{tr}, \phi) \in \mathsf{trace}_\Sigma(\mathcal{P})$ for some ϕ and (tr, ϕ) is quasi-well-typed w.r.t. $(\mathcal{T}_{\mathcal{P}}, \delta_{\mathcal{P}})$.

Intuitively, we can show that non equivalence relies on at most one disequality, and thanks to our equational theory, only two constants $\mathsf{c}^0_\star, \mathsf{c}^1_\star$ are necessary to produce such a disequality.

IV. From static equivalence to planning

The overall objective of this paper is to provide a practical algorithm for deciding trace equivalence, using planning graphs and SAT-solving. We start here with the static case and show how to reduce static equivalence to a planning problem. Given two frames, we show how to build a planning problem such that the planning problem has a solution if, and only if, the two corresponding frames are not in static equivalence.

We consider two frames ϕ and ψ having same domain. We denote Σ the constants from Σ_0 that occur either in ϕ or in ψ.

A. Planning problems

We first recall the definition of a planning problem, slightly simplified from [28]. Intuitively, a planning system defines a transition system from sets of facts to sets of facts. New facts may be produced and some old facts may be deleted.

Definition 8: A *planning system* is tuple $\langle \mathcal{F}act, \mathcal{I}nit, \mathcal{R}ule \rangle$ where $\mathcal{F}act$ is a set of variable-free atomic formulas called *facts*, $\mathcal{I}nit_0 \subseteq \mathcal{F}act$ is a set of facts representing the initial state, and $\mathcal{R}ule$ is a set of rules of the form:

$$Pre \rightarrow Add; Del$$

where Pre, Add, Del are finite sets of facts such that $Add \cap Del = \emptyset$, $Del \subseteq Pre$. We write $Pre \rightarrow Add$ when $Del = \emptyset$.

Given a rule $r \in \mathcal{R}ule$ of the form $Pre \rightarrow Add; Del$, we denote $Pre(r) = Pre$, $Add(r) = Add$, and $Del(r) = Del$. Moreover, if $S \subseteq \mathcal{F}act$ are such that $Pre(r) \subseteq S$, then we say that the rule is *applicable* in S, denoted $S \xrightarrow{r} S'$, and the state $S' = (S \setminus Del) \cup Add$ is the state resulting from the application of r to S. A *planning path* from $S_0 \subseteq \mathcal{F}act$ to $S_n \subseteq \mathcal{F}act$ is a sequence of rules $r_1, \ldots, r_n \in \mathcal{R}ule$ such that there exist states $S_1, \ldots, S_{n-1} \subseteq \mathcal{F}act$ such that:

$$S_0 \xrightarrow{r_1} S_1 \xrightarrow{r_2} \ldots S_{n-1} \xrightarrow{r_n} S_n$$

A planning problem for a system $\Theta = \langle \mathcal{F}act, \mathcal{I}nit, \mathcal{R}ule \rangle$ is a pair $\Pi = \langle \Theta, S_f \rangle$ where $S_f \subseteq \mathcal{F}act$ represents the target facts. A solution to $\Pi = \langle \Theta, S_f \rangle$, called a *plan*, is a planning path from $\mathcal{I}nit$ to a state S_n such that $S_f \subseteq S_n$.

In this paper, we consider an (infinite) set of facts $\mathcal{F}act_0$ that consists of:

- all atomic formulas of the form $\mathsf{att}(u_P, u_Q)$ with $u_P, u_Q \in \mathcal{M}_\Sigma$;
- all atomic formulas of the form $\mathsf{state}^c_{P,Q}(\sigma_P, \sigma_Q)$ where $c \in \mathcal{C}h$, P, Q are basic processes on channel c, and σ_P (resp. σ_Q) is a grounding substitution for P (resp. Q);
- a special symbol bad.

The rest of this section is dedicated to the reduction of static equivalence to the (non) existence of a solution of a planning system. Therefore, we will consider planning systems with facts that represent the attacker's knowledge, i.e. those of the form $\mathsf{att}(u_P, u_Q)$ (plus the symbol bad). Later on, in Section V, we will additionally consider the facts of the form $\mathsf{state}^c_{P,Q}(\sigma_P, \sigma_Q)$ that model internal states of the agents.

B. Attacker rules

We first describe the planning rules that correspond to the attacker behaviours. Instead of considering rules on ground facts, we start by describing a set of abstract rules that we

instantiated later on, yielding a (concrete) planning system. The attacker behaviour is modelled by the following set $\mathsf{Rule_A}$ of abstract rules:

$$\begin{array}{lcl}\mathsf{att}(\langle x_1, x_2\rangle, \langle y_1, y_2\rangle) & \to & \mathsf{att}(x_1, y_1)\\ \mathsf{att}(\langle x_1, x_2\rangle, \langle y_1, y_2\rangle) & \to & \mathsf{att}(x_2, y_2)\\ \mathsf{att}(\mathsf{enc}(x_1, x_2), \mathsf{enc}(y_1, y_2)),\ \mathsf{att}(x_2, y_2) & \to & \mathsf{att}(x_1, y_1)\end{array}$$

Note that there is no Del since the attacker never forgets. Interestingly, the rules only model decomposition There is no rule to synthesize messages. In general, this would be unsound but we will show why we can get rid of synthesis rules, thanks to the flattening technique. This is a key point of our algorithm to avoid building large terms.

We now explain how to obtain concrete planning rules from the abstract ones. This step is called concretization. Basically, we distinguish two kinds of concrete rules: the positive one, and the negative one. We start in this subsection by defining the positive one.

Given an abstract attacker rule $r \in \mathsf{Rule_A}$, we define its positive concretizations by simply instantiating the abstract rules such that the resulting terms are messages.

$$\mathsf{Concrete}^+(r) = \{r\sigma \mid \sigma \text{ a substitution grounding for } r \text{ such that } r\sigma \text{ only involve messages in } \mathcal{M}_\Sigma\}$$

Let ϕ and ψ be two frames with $dom(\phi) = dom(\psi)$. The set of facts associated to ϕ and ψ is defined as the set of couples of all identical constants and the couples of associated messages of the two frames.

$$\mathsf{Fact}(\phi, \psi) = \{\mathsf{att}(\mathsf{a}, \mathsf{a}) \mid a \in \Sigma\} \cup \{\mathsf{att}(\mathsf{w}\phi, \mathsf{w}\psi) \mid \mathsf{w} \in dom(\phi)\}$$

It is easy to show that, applying (concrete) attacker rules to $\mathsf{Fact}(\phi, \psi)$, we compute the set of couples (u, v) that can be reached by applying destructor-only recipes to ϕ and ψ.

Lemma 1: Let ϕ, ψ be two frames with $dom(\phi) = dom(\psi)$. Let $\Theta = \langle \mathcal{F}act_0, \mathsf{Fact}(\phi, \psi), \mathsf{Concrete}^+(\mathsf{Rule_A})\rangle$ and $\Pi = \langle \Theta, \{\mathsf{att}(u, v)\}\rangle$ for some $u, v \in \mathcal{M}_\Sigma$. We have that Π has a solution if, and only if, there is a destructor-only recipe $R \in \mathcal{R}_\Sigma$ such that $R\phi\downarrow = u$, and $R\psi\downarrow = v$.

C. Case of failures

To break static equivalence, an attacker may build new terms but also check for equalities and computation failures. Therefore, we encode when a computation can be performed on the right hand side but can not be mimicked on the left.

We say that a fact $f = \mathsf{att}(u_0, v_0)$ $(u_0, v_0 \in \mathcal{M}_\Sigma)$ *left-unifies* (resp. *right-unifies*) with $\mathsf{att}(u, v)$ if there exists σ such that $u\sigma = u_0$ (resp. $v\sigma = v_0$). Similarly, a sequence of facts $\mathsf{att}(u_1, v_1), \ldots, \mathsf{att}(u_k, v_k)$ left-unifies with a sequence $\mathsf{att}(u'_1, v'_1), \ldots, \mathsf{att}(u'_k, v'_k)$ if there exists σ such that $u'_1\sigma = u_1, \ldots, u'_k\sigma = u_k$ (and symmetrically for right-unification).

Given an abstract attacker rule $r = Pre \to Add$ (note that Del is empty for attacker rule), we define $\mathsf{Concrete}^-(r)$ as the set of concrete planning rules that contains:

$$f_1, \ldots, f_k \to \mathsf{bad}$$

for any sequence of facts $f_1, \ldots, f_k \in \mathcal{F}act_0$ such that $f_1, \ldots, f_k$ left-unifies with Pre, whereas $f_1, \ldots, f_k$ does not right-unify with Pre. This is the generic way to compute the failure rules from abstract attacker rules. In case of the set of abstract rules $\mathsf{Rule_A}$ that we consider here, we obtain the following infinite set of rules, denoted $\mathsf{Concrete}^-(\mathsf{Rule_A})$:

$$\mathsf{att}(\langle u_1, u_2\rangle, v) \to \mathsf{bad}$$
for any $u_1, u_2, v \in \mathcal{M}_\Sigma$ such that v is not a pair

$$\mathsf{att}(\mathsf{enc}(u_1, u_2), v), \mathsf{att}(u_2, v') \to \mathsf{bad}$$
for any $u_1, u_2, v, v' \in \mathcal{M}_\Sigma$ such that $\mathsf{enc}(u_1, u_2) \in \mathcal{M}_\Sigma$, and $\mathsf{dec}(v, v')\downarrow \notin \mathcal{M}_\Sigma$.

In order to capture static inclusion, we have to consider some additional cases of failure, in particular those corresponding to an equality that holds in one side but not in the other side. For this, we introduce the set $\mathcal{R}^{\mathsf{test}}_{\mathsf{fail}}$:

$$\mathcal{R}^{\mathsf{test}}_{\mathsf{fail}} = \{\mathsf{att}(u, v_1), \mathsf{att}(u, v_2) \to \mathsf{bad} \mid v_1 \neq v_2\}$$

However, as exemplified below, due to the absence of rule to compose terms, this is not sufficient.

Example 11: Let $\phi = \{\mathsf{w} \triangleright k\}$ and $\psi = \{\mathsf{w} \triangleright \mathsf{enc}(s, k)\}$ where $s, k \in \mathcal{N}$. We have that $\phi \not\sqsubseteq_s \psi$. Indeed, consider $R = \mathsf{enc}(\mathsf{w}, \mathsf{w})$, we have that $R\phi\downarrow \in \mathcal{M}_\Sigma$ whereas $R\psi\downarrow \notin \mathcal{M}_\Sigma$. However, we have no mean to witness this non-inclusion without relying on synthesis rules (that we do not have).

Therefore, we introduce in addition the ability to check whether a message is an atom or not (different from the special constant $\mathsf{c}_{\langle\omega,\omega\rangle}$). More formally, we consider the set:

$$\mathcal{R}^{\mathsf{atom}}_{\mathsf{fail}} = \{\mathsf{att}(u, v) \to \mathsf{bad} \mid u \text{ is an atom different from } \mathsf{c}_{\langle\omega,\omega\rangle} \text{ but not } v\}$$

Given a set Rule of abstract rules, we denote $\mathsf{Concrete}(\mathsf{Rule}) = \mathsf{Concrete}^+(\mathsf{Rule}) \cup \mathsf{Concrete}^-(\mathsf{Rule})$.

Two frames are in static inclusion if, and only if, the corresponding planning system has no solution.

Proposition 1: Let ϕ and ψ be two frames with $dom(\phi) = dom(\psi)$, and $\Theta = \langle \mathcal{F}act_0, \mathsf{Fact}(\phi, \psi), \mathcal{R}\rangle$ where

$$\mathcal{R} = \mathsf{Concrete}(\mathsf{Rule_A}) \cup \mathcal{R}^{\mathsf{test}}_{\mathsf{fail}} \cup \mathcal{R}^{\mathsf{atom}}_{\mathsf{fail}}.$$

Let $\Pi = \langle \Theta, \{\mathsf{bad}\}\rangle$. We have that $\phi \not\sqsubseteq_s \psi$ if, and only if, Π has a solution.

As we shall see later, in order to obtain an efficient algorithm, we do not enumerate all ground attacker rules. Instead, they are generated on the fly, only when they are needed.

V. From trace equivalence to planning

In the previous section, we have shown how to encode static inclusion into a planning system. We now show how to encode trace inclusion. We consider a protocol $\mathcal{P}$ that is type-compliant w.r.t. $(\mathcal{T}_\mathcal{P}, \delta_\mathcal{P})$, and another protocol $\mathcal{Q}$. For simplicity we assume that variables of $\mathcal{P}$ and $\mathcal{Q}$ are disjoint. Let $\Sigma = (\Sigma_\mathcal{P} \cup \Sigma_Q) \uplus \{\mathsf{c}^0_\star, \mathsf{c}^1_\star, \mathsf{c}_{\langle\omega,\omega\rangle}\}$. Moreover, we assume that variables occurring in $\mathcal{P}$ are given with types.

A. Protocol rules

We first define the abstract rules describing the protocol behaviour. Given P and Q two basic processes on channel c, we write $\mathsf{St}(P, Q) = \mathsf{state}^c_{P,Q}(id_P, id_Q)$ where id_P (resp. id_Q) is

the identity substitution of domain $fv(P)$ (resp. $fv(Q)$). Then, the transformation $\mathsf{Rule}(P;Q)$ from processes to abstract planning rules is defined as follows: We distinguish several cases depending on the shape of P.

1) Case $P = 0$:
$$\mathsf{Rule}(P;Q) = \emptyset.$$
2) Case $P = \mathsf{out}(c,u).P'$:
$$\mathsf{Rule}(P;Q) = \mathsf{Rule}(P';Q') \cup \{\mathsf{St}(P,Q) \to \mathsf{att}(u,v), \mathsf{St}(P',Q');\ \mathsf{St}(P,Q)\} \quad \text{when } Q = \mathsf{out}(c,v).Q'$$
$$\mathsf{Rule}(P;Q) = \{\mathsf{St}(P,Q) \to \mathsf{att}(u,\mathsf{c}_0^\star),\ \mathsf{bad}\} \quad \text{otherwise.}$$
3) Case $P = \mathsf{in}(c,u).P'$:
$$\mathsf{Rule}(P;Q) = \mathsf{Rule}(P';Q') \cup \{\mathsf{St}(P,Q), \mathsf{att}(u,v) \to \mathsf{St}(P',Q');\ \mathsf{St}(P,Q)\} \quad \text{when } Q = \mathsf{in}(c,v).Q'$$
$$\mathsf{Rule}(P;Q) = \{\mathsf{St}(P,Q), \mathsf{att}(u,x) \to \mathsf{bad}\} \quad \text{otherwise (with } x \text{ fresh).}$$

Intuitively, abstract rules simply try to mimic each step of P by a similar step of Q. Clearly, if Q cannot follow P, the two processes are not in trace equivalence, which is modelled here by the bad state. It then remains to check whether the bad state is indeed reachable. Note that, in case $P = \mathsf{out}(c,u).P'$ whereas Q is not ready to perform an output, bad will be trigger only if the outputted term is indeed a message.

Example 12: We consider protocols $\mathcal{P}'^1_{\mathsf{DS}}$ and $\mathcal{P}'^2_{\mathsf{DS}}$ as given in Example 7. We focus on the computations of the abstract protocol rules for the basic process defined on channel c_1, i.e.

$$\mathsf{Rule}(P_A^1.\mathsf{out}(c_1,\mathsf{enc}(\mathsf{m}_1,x_{AB}^1)), P_A^2.\mathsf{out}(c_1,\mathsf{enc}(\mathsf{m}_2,x_{AB}^2)))$$
where $P_A^i = \mathsf{out}(c_1,\langle \mathsf{a},\mathsf{b}\rangle).\ \mathsf{in}(c_1,\mathsf{enc}(\langle \mathsf{b},x_{AB}^i,x_B^i\rangle,k_{as})).\ \mathsf{out}(c_1,x_B^i).\ \mathsf{out}(c_1,\mathsf{enc}(\mathsf{m}_i,x_{AB}^i))$ with $i \in \{1,2\}$.

We have simply renamed bound variables to ensure disjointness between the variables of P_A^1 and those of P_A^2. Moreover, for sake of conciseness, below, we write $\mathsf{state}_i^{c_1}$ instead of $\mathsf{state}^{c_1}_{P_i^1,P_i^2}$ where P_i^1 (resp P_i^2) with $i \in \{1,4\}$ represents the subprocess of P_A^1 (resp. P_A^2) starting at the i^{th} action. We write id_X the identity substitution with $dom(id_X) = X$. Since this basic process is made up of 4 actions, we obtain 4 abstract protocol rules, among which the following abstract rule r_3:

$$\mathsf{state}_3^{c_1}(id_{\{x_{AB}^1,x_B^1\}}, id_{\{x_B^2\}}) \to \mathsf{att}(x_B^1,x_B^2), \mathsf{state}_4^{c_1}(id_{\{x_{AB}^1\}},\emptyset);\ \mathsf{state}_3^{c_1}(id_{\{x_{AB}^1,x_B^1\}}, id_{\{x_B^2\}})$$

Since both basic processes have the same shape, no absract rule with bad in conclusion have been computed at this stage.

This transformation is then extended to protocols in a natural way. Assume w.l.o.g. that both simple processes are made of n basic processes (we can complete with null processes if needed). That is, $\mathcal{P} = \{P_1,\ldots,P_n\}$ and $\mathcal{Q} = \{Q_1,\ldots,Q_n\}$. In addition, assume w.l.o.g. that P_i and Q_i are basic processes on channel c_i. We define

$$\mathsf{Rule}(\mathcal{P},\mathcal{Q}) = \mathsf{Rule}(P_1,Q_1) \cup \ldots \cup \mathsf{Rule}(P_n,Q_n).$$

Given a substitution σ, and $\mathsf{state}^c_{P,Q}(\sigma_P,\sigma_Q)$ occurring in a protocol abstract rule, the application of σ to the abstract state is defined as follows:

$$\mathsf{state}^c_{P,Q}(\sigma_P,\sigma_Q)\sigma = \mathsf{state}^c_{P,Q}(\sigma\circ\sigma_P, \sigma\circ\sigma_Q).$$

B. Flattening

In terms of efficiency, one key step of our algorithm is to avoid composition rules from the attacker. For this, we transform protocol rules in order to pre-compute all necessary composition steps. For example, consider the second step of the Denning Sacco protocol, presented in Example 4. The agent A expects a message m of the form $\{b,x_{AB},x_B\}_{k_{as}}$ and answers with x_B. Either the attacker obtains m as an existing ciphertext or he builds the ciphertext himself, provided he knows the key k_{as}. In the later case, we may avoid a composition step by considering the following (informal) rule:

$$b,\ x_{AB},\ x_B,\ k_{as} \ \to\ x_B$$

This rule is clearly useless for this particular example but illustrates our flattening technique. Note that such rules will become useful for the analysis of a more complex scenario, in particular those involving dishonest participants.

We now explain how formally to compute the set of flattened rules from a given abstract rule r. For this, we start by explaining how to decompose a fact $\mathsf{att}(u,v)$.

Definition 9: Given a term $u \in \mathcal{T}(\Sigma_c, \Sigma\cup\mathcal{N}\cup\mathcal{X})$, we say that u is *decomposable* when:

- either $u \in \mathcal{X}$ and $\delta_{\mathcal{P}}(u)$ is not an atomic type;
- or $u \notin \Sigma\cup\mathcal{N}\cup\mathcal{X}$.

Intuitively, a variable of non atomic type is decomposable since it may be instantiated by a non atomic term which, in turns, may have been obtained by composition. Given $\mathsf{att}(u,v)$ with u decomposable, we define $\mathsf{split}(\mathsf{att}(u,v))$ as follows:

$$\mathsf{split}(\mathsf{att}(u,v)) = (\mathsf{f};\{\mathsf{att}(x_1,y_1),\mathsf{att}(x_2,y_2)\};\sigma_{\mathcal{P}};\sigma_{\mathcal{Q}})$$

where

- $\delta_{\mathcal{P}}(u) = \mathsf{f}(\tau_1,\tau_2)$ for some τ_1,τ_2 and some $\mathsf{f} \in \Sigma_c$;
- x_1 (resp. x_2) is a fresh variable of type τ_1 (resp. τ_2) and $\sigma_{\mathcal{P}} = \mathsf{mgu}(u,\mathsf{f}(x_1,x_2))$;
- y_1,y_2 are fresh variables, $\sigma_{\mathcal{Q}} = \mathsf{mgu}(v,\mathsf{f}(y_1,y_2))$.

Note that $\sigma_{\mathcal{P}}$ exists and is necessarily a well-typed substitution. By convention, we assume that $\mathsf{mgu}(u,u') = \bot$ when u and u' are not unifiable.

Let r be an abstract rule of the form $Pre \to Add; Del$ with $f = \mathsf{att}(u,v) \in Pre$ such that u is decomposable and $\mathsf{split}(f) = (\mathsf{f},S,\sigma_{\mathcal{P}},\sigma_{\mathcal{Q}})$. The decomposition of r w.r.t. f, denoted $\mathsf{decompo}(r,f)$, is defined as follows:

1) $\big((Pre \setminus f)\cup S \to \mathsf{bad}\big)\sigma_{\mathcal{P}}$ in case $\sigma_{\mathcal{Q}} = \bot$;
2) $\big((Pre \setminus f)\cup S \to Add; Del\big)(\sigma_{\mathcal{P}} \uplus \sigma_{\mathcal{Q}})$ otherwise.

Then, decomposition is applied recursively on each rule.

$$\mathsf{Flat}(r) = \mathsf{Flat}(\{\mathsf{decompo}(r,f) \mid f = \mathsf{att}(u,v) \in Pre(r) \text{ with } u \text{ decomposable}\}) \cup \{r\}$$

Example 13: Considering the abstract protocol rule r_3 given in Example 12, the set $\mathsf{Flat}(r_3)$ contains (among others):

$$\begin{array}{l}\mathsf{state}_2^{c_1}(\emptyset,\emptyset),\\ \mathsf{att}(\langle \mathsf{b}, x^1_{AB}, x^1_B\rangle, \langle \mathsf{b}, x^2_{AB}, x^2_B\rangle), \mathsf{att}(k_{as}, k_{as})\\ \qquad \rightarrow \mathsf{state}_3^{c_1}(id_{\{x^1_{AB}, x^1_B\}}, id_{\{x^2_B\}});\ \mathsf{state}_2^{c_1}(\emptyset,\emptyset)\end{array}$$

$$\begin{array}{l}\mathsf{state}_2^{c_1}(\emptyset,\emptyset),\\ \mathsf{att}(\mathsf{b},\mathsf{b}), \mathsf{att}(x^1_{AB}, x^2_{AB}), \mathsf{att}(x^1_B, x^2_B), \mathsf{att}(k_{as}, k_{as})\\ \qquad \rightarrow \mathsf{state}_3^{c_1}(id_{\{x^1_{AB}, x^1_B\}}, id_{\{x^2_B\}});\ \mathsf{state}_2^{c_1}(\emptyset,\emptyset)\end{array}$$

$$\begin{array}{l}\mathsf{state}_2^{c_1}(\emptyset,\emptyset), \mathsf{att}(\mathsf{b},\mathsf{b}), \mathsf{att}(x^1_{AB}, x^2_{AB}), \mathsf{att}(k_{as}, k_{as})\\ \mathsf{att}(x^1_{B1}, x^2_{B1}), \mathsf{att}(x^1_{B2}, x^2_{B2})\\ \qquad \rightarrow \mathsf{state}_3^{c_1}(\sigma_1, \sigma_2);\ \mathsf{state}_2^{c_1}(\emptyset,\emptyset)\end{array}$$

where

- $\sigma_1 = \{x^1_{AB} \mapsto x^1_{AB}, x^1_B \mapsto \mathsf{enc}(x^1_{B1}, x^1_{B2})\}$;
- $\sigma_2 = \{x^2_B \mapsto \mathsf{enc}(x^2_{B1}, x^2_{B2})\}$; and
- x^1_{B1} (resp. x^2_{B2}) is of type $\langle \tau_k, \tau_{\mathsf{a}}\rangle$ (resp. τ_{ks}).

C. Concretization

Given an abstract rule r, we denote $vars_{\mathsf{left}}(r)$ the variables occurring on the left (first parameter) of a predicate occuring in r, and similarly for $vars_{\mathsf{right}}(r)$. More precisely,

- $vars_{\mathsf{left}}(\mathsf{att}(u,v)) = vars(u)$; and
- $vars_{\mathsf{left}}(\mathsf{state}^c_{P,Q}(\sigma_P,\sigma_Q)) = vars(img(\sigma_P))$.

We have that $vars(r) = vars_{\mathsf{left}}(r) \uplus vars_{\mathsf{right}}(r)$.

Given an abstract protocol rule r, its positive concretization simply consists in all its instantiations that are well-typed w.r.t. the left side of the rule.

$$\begin{array}{l}\mathsf{Concrete}^+(r) = \{r\sigma \mid\ \sigma \text{ a substitution grounding for } r\\ \qquad\qquad \text{such that } r\sigma \text{ only involve messages in } \mathcal{M}_\Sigma\\ \qquad\qquad \text{and } \delta_{\mathcal{P}}(x\sigma) \preceq \delta_{\mathcal{P}}(x) \text{ for any } x \in vars_{\mathsf{left}}(r)\}\end{array}$$

Let $K_P = (\mathcal{P}; \sigma_{\mathcal{P}}; \phi)$ and $K_Q = (\mathcal{Q}; \sigma_{\mathcal{Q}}; \psi)$ be two configurations with $dom(\phi) = dom(\psi)$. The set of facts associated to K_P and K_Q is defined as follows:

$$\begin{array}{l}\mathsf{Fact}(K_P, K_Q) \ = \ \mathsf{Fact}(\phi,\psi) \cup\\ \quad \{\mathsf{state}^c_{P,Q}(\sigma_P,\sigma_Q) \mid P \in \mathcal{P}, Q \in \mathcal{Q} \text{ are basic processes}\\ \qquad \text{on channel } c,\ \sigma_P = \sigma_{\mathcal{P}}|_{fv(P)} \text{ and } \sigma_Q = \sigma_{\mathcal{Q}}|_{fv(Q)}\ \}\end{array}$$

We denote by $\mathsf{Fact}(K_P, K_Q) \uparrow S'$ when the set of facts S' can be obtained from the set of facts $\mathsf{Fact}(K_P, K_Q)$ by adding only deducible facts (using destructor recipes only).

Definition 10: Given two sets of facts S and S' such that $S = \mathsf{Fact}(K_P, K_Q)$ with $K_P = (\mathcal{P}; \phi; \sigma_P)$ and $K_Q = (\mathcal{Q}; \psi; \sigma_Q)$ with $dom(\phi) = dom(\psi)$, we write $\mathsf{Fact}(K_P, K_Q) \uparrow S'$ when:

- $\mathsf{Fact}(K_P, K_Q)$ and S' coincide on states;
- for any $\mathsf{att}(u,v) \in \mathsf{Fact}(K_P, K_Q)$, $\mathsf{att}(u,v) \in S'$; and
- for any $\mathsf{att}(u,v) \in S'$, there exists a destructor-only recipe R such that $R\phi\downarrow = u$, and $R\psi\downarrow = v$.

The solutions of the planning system obtained as the positive concretization of the abstract rules of P and Q exactly corresponds to the set of (quasi-well-typed) traces of P that can be mimicked by Q.

Lemma 2: Let $\mathcal{P}$ be a protocol type-compliant w.r.t. $(\mathcal{T}_{\mathcal{P}}, \delta_{\mathcal{P}})$, and $\mathcal{Q}$ be another protocol. Let Θ be the following planning system:

$$\langle \mathcal{F}act_0, \mathsf{Fact}(\mathcal{P},\mathcal{Q}), \mathcal{R}\rangle$$

where $\mathcal{R} = \mathsf{Concrete}^+(\mathsf{Rule}_{\mathsf{A}} \cup \mathsf{Flat}(\mathsf{Rule}(\mathcal{P},\mathcal{Q})))$.

Let $(\mathsf{tr}, \phi) \in \mathsf{trace}_\Sigma(P)$ for some ϕ and such that:

- tr only contains simple recipes;
- (tr, ϕ) is well-typed w.r.t. $(\mathcal{T}_{\mathcal{P}}, \delta_{\mathcal{P}})$;
- $(\mathsf{tr}, \psi) \in \mathsf{trace}_\Sigma(\mathcal{Q})$ for some ψ.

Then, there exist a planning path $r_1, \ldots, r_n$ of some length n from $\mathsf{Fact}(\mathcal{P},\mathcal{Q})$ to some S_n such that $\mathsf{Fact}(K'_P, K'_Q) \uparrow S_n$ where K'_P (resp. K'_Q) is the resulting configuration starting from $\mathcal{P}$ (resp. $\mathcal{Q}$) and executing tr.

Conversely, let $r_1, \ldots, r_n$ be a planning path from $\mathsf{Fact}(\mathcal{P},\mathcal{Q})$ to S_n such that $\mathsf{bad} \notin S_n$. Then, there exist a trace tr, and frames ϕ and ψ such that:

- tr only contains simple recipes;
- (tr, ϕ) is well-typed w.r.t. $(\mathcal{T}_{\mathcal{P}}, \delta_{\mathcal{P}})$;
- $(\mathsf{tr}, \psi) \in \mathsf{trace}_\Sigma(\mathcal{Q})$ for some ψ; and
- $\mathsf{Fact}(K'_P, K'_Q) \uparrow S_n$ where K'_P (resp. K'_Q) is the resulting configuration starting from $\mathcal{P}$ (resp. $\mathcal{Q}$) and executing tr.

D. Case of failures

Similarly to the static case, we need to make sure that we can detect when P and Q are *not* in trace inclusion. For this, we consider additional rules that express when a step that can be performed on the left hand side cannot be mimicked on the right hand side.

Given an abstract protocol rule $r = Pre \rightarrow Add; Del$, $\mathsf{Concrete}^-(r)$ is the set of planning rules that contains:

$$f_1, \ldots, f_k \rightarrow \mathsf{bad}$$

for any sequence of facts $f_1, \ldots, f_k \in \mathcal{F}act_0$ such that $f_1, \ldots, f_k$ left-unify with Pre with substitution σ_L and $u \in \mathcal{M}_\Sigma$ for any $\mathsf{att}(u,v) \in Add\sigma_L$, and such that one of the following conditions holds:

- $f_1, \ldots, f_k$ does not right-unify with Pre;
- $f_1, \ldots, f_k$ right-unify with Pre with substitution σ_R but $v \notin \mathcal{M}_\Sigma$ for some $\mathsf{att}(u,v) \in Add\sigma_R$.

Our main technical result states that our encoding in planning system is sound and complete: two protocols are in trace inclusion if, and only if, the corresponding planning system (obtained by considering both positive and negative concretizations) has a solution.

Theorem 3: Let $\mathcal{P}$ a protocol type-compliant w.r.t. $(\mathcal{T}_{\mathcal{P}}, \delta_{\mathcal{P}})$, and $\mathcal{Q}$ be another protocol. We consider the following set $\mathcal{R}$ of concrete rules:

$$\mathcal{R} = \mathsf{Concrete}(\mathsf{Rule}_{\mathsf{A}} \cup \mathsf{flat}(\mathsf{Rule}(\mathcal{P},\mathcal{Q}))) \cup \mathcal{R}^{\mathsf{test}}_{\mathsf{fail}} \cup \mathcal{R}^{\mathsf{atom}}_{\mathsf{fail}}$$

Let $\Theta = \langle \mathcal{F}act_0, \mathsf{Fact}(\mathcal{P},\mathcal{Q}), \mathcal{R}\rangle$ and $\Pi = \langle \Theta, \{\mathsf{bad}\}\rangle$. We have that $\mathcal{P} \not\sqsubseteq \mathcal{Q}$ if, and only if, Π has a solution.

This reduction to a planning system is a key ingredient of our result. But of course, it does not immediately yields an algorithm since the planning system encoding trace inclusion of a process $\mathcal{P}$ w.r.t. a process $\mathcal{Q}$ is actually infinite. Indeed, consider for example the positive concretizations of an abstract rule in $\mathsf{Rule}(\mathcal{P};\mathcal{Q})$. There are finitely many instantiations for the "left" part, that corresponds to $\mathcal{Q}$ thanks to the typing system. However, the "right" part (corresponding to $\mathcal{Q}$) may be instantiated arbitrarily. We explain how to design an (efficient) algorithm in the next section.

VI. Algorithm

Our algorithm takes as input a protocol $\mathcal{P}$ that is type-compliant w.r.t. a typing system $(\mathcal{T}_\mathcal{P}, \delta_\mathcal{P})$ and another protocol $\mathcal{Q}$. We explain here how to check trace inclusion of $\mathcal{P}$ in protocol $\mathcal{Q}$. Then, trace equivalence is obtained by checking trace inclusion of $\mathcal{P}$ in $\mathcal{Q}$ and $\mathcal{Q}$ in $\mathcal{P}$.

Step 1: Compute the abstract rules of $(\mathcal{P};\mathcal{Q})$. As explained in Section V-A, we compute the abstract rules $\mathsf{Rule}(\mathcal{P};\mathcal{Q})$ associated to $(\mathcal{P};\mathcal{Q})$, and then their flattened version $\mathsf{flat}(\mathsf{Rule}(\mathcal{P};\mathcal{Q}))$, as described in Section V-B. Together with the attacker rules (defined in Section IV-B), this yields

$$\mathsf{Rule}_\mathsf{A} \cup \mathsf{flat}(\mathsf{Rule}(\mathcal{P};\mathcal{Q})).$$

Step 2: Initial state. Thanks to Theorem 2, it is sufficient to consider at most three extra constants in addition to the constants of $\mathcal{P}$ and $\mathcal{Q}$, that is, it is sufficient to consider $\Sigma = \Sigma_\mathcal{P} \cup \Sigma_\mathcal{Q} \uplus \{\mathsf{c}^0_\star, \mathsf{c}^1_\star, \mathsf{c}_{\langle\omega,\omega\rangle}\}$. We then add the initial states of the protocols. More formally, we start with the initial state

$$\mathsf{Fact}(\mathcal{P},\mathcal{Q}).$$

Step 3: Planning graph algorithm. Given a planning system, the standard technique for finding a solution to the planning system is to apply the planning graph algorithm (see [14]), that we briefly recall here. The algorithm consists in building a graph (called planning graph), that consists in an alternance of facts layers and rules layers, linked with four kinds of edges: *Pre*, *Add* and *Del* edges, that are edges between the fact layers and the rule layers; and mutex (as in *mutual exclusion*) are edges between vertices of the same layers. Mutex edges indicate when vertices may not be obtained simultaneously.

More precisely, the planning graph algorithm proceeds as follows. Let i denote the number of layers. Initially, $i := 0$.

1) The first fact layer is $N^f_0 = \mathsf{Fact}(\mathcal{P},\mathcal{Q})$ (the set of initial facts) and the first rule layer is empty, $N^r_0 = \emptyset$.
2) From the fact layer N^f_i, compute the set R of all concrete rules (either from $\mathsf{Concrete}^+$ or $\mathsf{Concrete}^-$) that are applicable from N^f_i without any mutex edge between facts of their precondition. Since there are a finite number of abstract protocol rules and since the facts in N^f_i are ground and finite, the set R of concrete rules applicable from N^f_i is finite as well.
3) Compute the new mutex edges between the rules. Rules are in mutex if they either interfere (one deletes a precondition or an add-effect of the other) or have competing needs (there is a mutex edge between their preconditions).
4) Build N^f_{i+1} from N^r_i by adding the facts introduced by the rules in N^r_i. We have that:
$$N^f_{i+1} = \cup_{\rho\in N^r_i}$$
5) We compute the mutex edges between facts. There is a mutex edge between two facts f, f' if each rule that adds f is in mutex with each rule that adds f'.
6) $i := i + 1$
7) Check whether $N^f_i := N^f_{i-1}$ (same facts and same mutex). If yes, then stop. Otherwise, go back to Point 2.

When the planning graph algorithm stops, we obtain a graph, that represents an over-approximation of the states reachable from the planning system, starting from the initial state. While we are looking for a solution to an infinite planning system (finitely described through abstract rules), we only need to consider a finite number of concrete rules at each round of the algorithm (Point 2 of the algorithm). Note that this construction is not a naive saturation that would explore all possible paths. The mutex edges ensure a not too coarse over-approximation and provide a mean for considering the application of a rule to a family of facts instead a single fact.

Step 4: SAT encoding. If bad does not occur in the resulting planning graph, then trace inclusion is guaranteed since the planning graph is an over-approximation of the reachable states. If bad does occur in the planning graph, we can check whether bad is indeed reachable through SAT solvers. More precisely, we encode the existence of a solution to the planning system into a SAT instance, using the same technique as SATMC (see [16]), and relying on the SAT solver minisat [29]. If bad is reachable, the SAT solver provides us with a solution, which is translated back to an attack trace. If bad is not reachable (that is, the SAT solver guarantees that there is no solution), then trace inclusion is guaranteed.

Conclusion. Thanks to Theorem 3, $\mathcal{P} \not\sqsubseteq_t \mathcal{Q}$ if, and only if, the corresponding planning system $\mathcal{R}$ has a solution, that is, bad is reachable. Therefore our algorithm is correct and complete: it provides an attack if, and only if, $\mathcal{P} \not\sqsubseteq_t \mathcal{Q}$. Since $\mathcal{P} \approx_t \mathcal{Q}$ is defined as $\mathcal{P} \sqsubseteq_t \mathcal{Q}$ and $\mathcal{Q} \sqsubseteq_t \mathcal{P}$, we can then easily check whether two processes are in trace equivalence ($\mathcal{P} \approx_t \mathcal{Q}$).

Termination. Our procedure is not guaranteed to terminate. This may be surprising since Theorem 1 ensures that it is sufficient to consider traces that are well-types w.r.t. $(\mathcal{T}_\mathcal{P}, \delta_\mathcal{P})$. Then, since the processes are deterministic, a given trace of $\mathcal{P}$ can only be followed by at most one trace in $\mathcal{Q}$, hence a finite number of traces need to be considered. However, the planning graph step over-approximates the set of facts that need to be considered. Therefore, it may consider several facts of the form $\mathsf{att}(u,u_1)$, $\mathsf{att}(u,u_2)$, ..., $\mathsf{att}(u,u_n)$ with distinct, uncontrolled, u_i. One way to enforce termination would be to check at each step that the planning graph only considers reachable facts (applying our SAT encoding). However, this would considerably slow down our algorithm while our experiments show that, not only our algorithm

terminates in practice, but it is also much more efficient than other existing tools.

VII. Case studies

In this section, we analyse several protocols of the literature and compare the results obtained using different tools that decide equivalence for a bounded number of sessions. The characteristics of these tools are given in Section VII-A, the different scenarios including some scenarios with corruption are described in Section VII-B. The result are described In Section VII-C and a discussion is provided in Section VII-D. Our tool as well as the source files to reproduce the benchmarks are available at [20].

A. Tools

Spec [9], [30] deals with a fixed set of cryptographic primitives, namely symmetric encryption and pairs, and protocols with no else branch. The procedure is sound and complete w.r.t. open bisimulation (a notion that is strictly stronger than trace equivalence [31]) and its termination is proved [9].

Apte decides trace equivalence [8], [11], [32] for a fixed but richer set of cryptographic primitives (i.e. symmetric/asymmetric encryptions, signature, pair, and hash functions). Processes are also more general: they include private channels, internal communications, processes that are not necessarily simple, and possibly with else branches.

Systems we are interested in are highly concurrent and a naive exploration of all possible interleavings limits the practical impact of those tools. Recent works [12], [13] have partially addressed the state space explosion problem due to naive exploration of all possible interleavings implemented in this tool. These dedicated partial order reduction (POR) techniques have been implemented in **Apte-por** (as an option of the Apte tool) yielding a significant speed-up.

Akiss implements the procedure described in [10], [33] and deals with rich user-defined term algebras including symmetric encryption and pairs. It is able to check an over-approximation of trace equivalence that actually coincides with trace equivalence for the class of simple processes that we study in this paper. Its termination has been established for the particular set of primitives used in this paper [34], and the perfomance of the tool has been recently improved relying on POR techniques mentionned above.

Of course, not all the tools consider exactly the same semantics. For example, Akiss considers a true equational theory while Spec, Akiss, and SAT-equiv consider a rewrite system (with again subtle differences). We believe nevertheless that they prove very similar properties and we therefore compare here their performance.

B. Scenarios with corruption

The scenario we considered so far for the Denning-Sacco protocol is quite simple. We only consider two sessions involving honest agents. This scenario involves 6 roles in parallel, and is denoted DS-6 in the table given in Section VII-C.

In the same spirit, we consider a simpler scenario, denoted DS-3, that corresponds to only one instance of each role (between honest agents). Such scenarios are known to be too simplistic and some attacks may be missed.

To go furhter, we consider scenario where honest agents are willing to engage communications with a dishonest agent c. Let us develop this corruption scenario on the Denning-Sacco protocol. Formally, we consider in addition of the three basic processes used to model scenario DS-3, a basic process to model that the agent a may be involved in another session with a corrupted agent c, and the server S is ready to answer a request coming from them. Similary, we consider also two additional basic processes to model the fact that agent b may be involved in another session where the role A is played by the corrupted agent c. This scenario is therefore made up of 7 basic processes and is named accordingly DS-7.

To be more complete, we can also consider the cases where the role of A is played by b, and the role of B is played by a (scenario DS-10), and then we add again processes to model sessions with a corrupted agent (scenario DS-12 and DS-14).

We consider the case where the property is encoded on role B (strong secrecy of the key as received by B). We may also decide to encode the property on the two instances of the roles of B (scenario DS-6-bis) or only once (scenario DS-6).

C. Review of symmetric key protocols

Most of the protocols we considered from [24] actually fall in our class. We sometimes need to include some explicit tags to ensure type-compliance (this check is performed automatically by our tool). We now report on experimental results. We ran the different tools on a single Intel 3.1 GHz Xeon core with 190Go of RAM (shared with the other 19 cores) and we compare their performances on several protocols. For SAT-Equiv, we further indicate the number of ground facts and rules considered when computing the planning graph.

We decide to stop each experiment after 24h, and we indicate by TO (Time Out) when the tool does not return an answer within this timeframe, SO when we encounter a stack overflow, and MO in case the tool used more than 64 Go of Memory. We encountered some bugs that are indicated by BUG when interacting with Apte (internal errors or wrong results). We have reported these bugs to the authors.

Some protocols are subject to replay attacks, detected by the scenario 6-bis mentioned earlier. Even if scenarios that correspond to an attack are less interesting regarding performances comparison (since most of the tools stop their exploration once an attack has been found), we report the corresponding analysis in the last raw of each table, whenever applicable, that is, whenever there is indeed an attack.

Denning Sacco. The Denning Sacco protocol has been described in Example 4. There is a replay attack on DS-6-bis due to a lack of freshness on the messages that are exchanged. This attack is similar to the one explained in Example 7.

DS	Spec	Akiss	Apte	Apte-por	Sat-Eq	
3	12s	0.10s	0.3s	0.03s	0.25s	58
6	MO	15s	TO	8s	1s	104
7		101s		13s	2s	132
10		SO		39m	4s	166
12				TO	7s	203
14					10s	234
6-bis	78m	49s	19s	0.07s	2s	122

Wide Mouth Frog. We consider the protocol as described in [24] but without timestamps as described below:

$$\begin{array}{ll} A \to S: & A,\ \{B, K_{ab}\}_{K_{as}} \\ S \to B: & \{A, K_{ab}\}_{K_{bs}} \end{array}$$

Therefore, there is a replay attack on WMF-6-bis due to a lack of freshness on the messages that are exchanged.

WMF	Spec	Akiss	Apte	Apte-por	Sat-Eq	
3	6s	0.04s	0.06s	0.01s	0.10s	52
6	58m	1.6s	55m	1.5s	1s	96
7	TO	5.3s	TO	2s	2s	121
10		8m30s		22m	7s	165
12		SO		TO	40s	238
14					118s	312
6-bis	13m	5.7s	0.06s	0.06s	1s	114

Needham-Schroeder. We consider the Needham-Schroeder protocol based on symmetric encryption as described in [24] (see below).

$$\begin{array}{ll} A \to S: & A, B, N_a \\ S \to A: & \{B, N_a, K_{ab}, \{A, K_{ab}\}K_{bs}\}_{K_{as}} \\ A \to B: & \{A, K_{ab}\}_{K_{bs}} \\ B \to A: & \{Req, N_b\}_{K_{ab}} \\ A \to B: & \{Rep, N_b\}_{K_{ab}} \end{array}$$

NS	Spec	Akiss	Apte	Apte-por	Sat-Eq	
3	63s	4.4s	0.4s	0.03s	2s	100
6	MO	TO	TO	11m	54s	245
7				TO	153s	342
10					8m	475
12					22m	622
14					77m	838

Yahalom-Lowe. We consider the protocol as described in [24]. However, to ensure type-compliance, we consider a tagged version of the protocol.

$$\begin{array}{ll} A \to B: & A, N_a \\ B \to S: & \{1, A, N_a, N_b\}_{K_{bs}} \\ S \to A: & \{2, B, K_{ab}, N_a, N_b\}_{K_{as}} \\ S \to B: & \{3, A, K_{ab}\}_{K_{bs}} \\ A \to B: & \{4, A, B, S, N_b\}_{K_{ab}} \end{array}$$

YL	Spec	Akiss	Apte	Apte-por	Sat-Eq	
3	11s	3s	12s	0.12s	5s	122
6	MO	TO	TO	35m	3m	333
7				BUG	19m	549
10					206m	934
12					19h	1391
14					TO	_

Yahalom-Paulson. We consider the protocol as described in [24]. To ensure type-compliance, we consider a tagged version of the protocol.

$$\begin{array}{ll} A \to B: & A, N_a \\ B \to S: & B, N_b, \{1, A, N_a\}_{K_{bs}} \\ S \to A: & N_b, \{2, B, K_{ab}, N_a\}_{K_{as}}, \{3, A, B, K_{ab}, N_b\}_{K_{bs}} \\ A \to B: & \{3, A, B, K_{ab}, N_b\}_{K_{bs}}, \{4, N_b\}_{K_{ab}} \end{array}$$

YP	Spec	Akiss	Apte	Apte-por	Sat-Eq	
3	23m	7s	111s	0.9s	50s	234
6	MO	TO	TO	BUG	165m	976
7					TO	_

Otway-Rees. We have also analysed a tagged version of the Otway-Rees protocol (see [24]).

$$\begin{array}{ll} A \to B: & M, A, B, \{1, N_a, M, A, B\}K_{as} \\ B \to S: & M, A, B, \{1, N_a, M, A, B\}K_{as}, \\ & \qquad \{2, N_b, M, A, B\}K_{bs} \\ S \to B: & M, \{3, N_a, K_{ab}\}K_{as}, \{4, N_b, K_{ab}\}K_{bs} \\ B \to A: & M, \{3, N_a, K_{ab}\}K_{as} \end{array}$$

OR	Spec	Akiss	Apte	Apte-por	Sat-Eq	
3	16m	225s	BUG	24s	104s	239
6	MO	SO		SO	46m	660
7					50m	637
10					276m	1033
12					9h40m	1265
14					TO	_

Simple stateful example. Some protocols are stateful (see [35] for a detailed discussion). For example, a process may lock a ressource which cannot be used until it is unlocked. We consider here a mock protocol that reflects this type of behaviors. The protocol $P_{\mathsf{yes}}(n)$ with n tokens is described informally below ($1 \leq i \leq n$), and is made of $3n$ processes running in parallel on distinct channels.

$$\begin{array}{lrcl} 1. & & \to & \{toka_i\}_{k_i}, \{tokb_i\}_{k_i} \\ 2. & \{x\}_{k_i} & \to & x \\ 3. & toka_i, tokb_i & \to & \mathsf{yes} \end{array}$$

Here, yes and no are public constants, whereas k_i, $toka_i$, and $tokb_i$ are names unknown by the attacker. The protocol $P_{\mathsf{no}}(n)$ can be defined similarly. Intuitively, $P_{\mathsf{yes}}(n) \approx P_{\mathsf{no}}(n)$ holds since rule 2 can be used only once for each key k_i. Therefore,

it is never possible to trigger a rule of type 3. We checked equivalence using ProVerif and, unsurprinsgly, it found a false attack. This is due to the fact that ProVerif cannot properly model "a finite amount of time".

# tok.	Spec	Akiss	Apte	Apte-por	Sat-Eq	
1	15s	0.02s	0.09s	0.01s	0.16s	49
2	MO	0.37s	240m	0.15s	1s	100
3		18s	MO	5s	2s	144
4		SO		9min32s	6s	188
12				TO	155s	540
36					85m	1596
60					6h40m	2652

D. Discussion

For ease of comparison, we decided to run our experiments using a single core machine since not all the tools are able to take advantage of more cores. Running these examples using more cores would have benefited to our tool that reaches its optimum when it is launched using 4 cores (2 inclusions have to be checked with constants $\mathsf{c}_\star^0$ and $\mathsf{c}_\star^1$ (resp. $\mathsf{c}_{\langle\omega,\omega\rangle}$)), and also to Akiss on which the saturation process is highly parallelizable.

The obtained results give evidence that our technique is less sensitive to the number of concurrent sessions analysed. On the contrary, the other tools that handle messages symbolically are less sensitive to the size of messages, which explains why our tool is typically slower on a small number of sessions. Moreover, on all our secure examples on which no attack is found, the planning graph is an over-approximation that appears to be precise enough, and does not require calls to the SAT solver. For the examples where an attack has been found (DS-6-bis and WMF-6-bis), the resulting SAT formulas contain about 750 variables and 4000 clauses.

VIII. Conclusion

Our tool SAT-Equiv outperforms all existing tools, even for the new Apte-por variant of Apte and the recently updated Akiss tool on which POR techniques have also been integrated. We also discovered several bugs in Apte-por, which prevented us from a thorough comparison of the two tools. SAT-Equiv is sometimes slower for a small number of sessions but in all cases, SAT-Equiv is the tool that allows to analyze the largest number of sessions.

One limitation of our tool is the fact that it covers protocols with symmetric encryption only. This is not an intrinsic limitation of our approach but rather a current limitation of the typing result [19], which states that we can limit ourselves to well-type attack traces. We plan to extend [19] to all standard primitives and we believe that the extension to SAT-Equiv to all standard primitives would then follow quite easily.

Note also that our tool is not guaranteed to terminate. We could enforce termination by checking reachability of the considered facts while building the planning graph, at the price of considerably slowing down our tool. Instead, as future work, we plan to formally prove termination of the planning graph construction or to identify under which assumptions, termination can be guaranteed.

Acknowledgments

This work has been partially supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreements No 645865-SPOOC and No 714955-POPSTAR) and the DGA.

REFERENCES

[1] B. Blanchet, "An Efficient Cryptographic Protocol Verifier Based on Prolog Rules," in *14th IEEE Computer Security Foundations Workshop (CSFW-14)*. Cape Breton, Nova Scotia, Canada: IEEE Computer Society, Jun. 2001, pp. 82–96.

[2] A. Armando *et al.*, "The AVANTSSAR platform for the automated validation of trust and security of service-oriented architectures," in *Proceedings of the 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'2012), 2012, Tallinn, Estonia, March 24 - April 1, 2012.*, ser. Lecture Notes in Computer Science, C. Flanagan and B. König, Eds., vol. 7214. Springer, 2012, pp. 267–282. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-28756-5_19

[3] S. Escobar, C. Meadows, and J. Meseguer, "A rewriting-based inference system for the NRL protocol analyzer and its meta-logical properties," *Theoretical Computer Science*, vol. 367, no. 1-2, pp. 162–202, 2006.

[4] C. Cremers, "The Scyther Tool: Verification, falsification, and analysis of security protocols," in *Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, USA, Proc.*, ser. Lecture Notes in Computer Science, vol. 5123/2008. Springer, 2008, pp. 414–418.

[5] S. Meier, B. Schmidt, C. Cremers, and D. Basin, "The TAMARIN Prover for the Symbolic Analysis of Security Protocols," in *Computer Aided Verification, 25th International Conference, CAV 2013, Princeton, USA, Proc.*, ser. Lecture Notes in Computer Science, N. Sharygina and H. Veith, Eds., vol. 8044. Springer, 2013, pp. 696–701.

[6] S. Santiago, S. Escobar, C. A. Meadows, and J. Meseguer, "A Formal Definition of Protocol Indistinguishability and Its Verification Using Maude-NPA," in *STM 2014*, ser. LNCS, 2014, pp. 162–177.

[7] D. L. Mitchell, N. A. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov, "Undecidability of bounded security protocols," 1999.

[8] V. Cheval, H. Comon-Lundh, and S. Delaune, "Trace equivalence decision: Negative tests and non-determinism," in *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS'11)*. Chicago, Illinois, USA: ACM Press, Oct. 2011. [Online]. Available: http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/CCD-ccs11.pdf

[9] J. Dawson and A. Tiu, "Automating open bisimulation checking for the spi-calculus," in *Proceedings of IEEE Computer Security Foundations Symposium (CSF 2010)*, 2010.

[10] R. Chadha, Ş. Ciobâcă, and S. Kremer, "Automated verification of equivalence properties of cryptographic protocols," in *Programming Languages and Systems —Proceedings of the 21th European Symposium on Programming (ESOP'12)*, ser. Lecture Notes in Computer Science, H. Seidl, Ed., vol. 7211. Tallinn, Estonia: Springer, Mar. 2012, pp. 108–127.

[11] V. Cheval, "Apte: an algorithm for proving trace equivalence," in *Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14)*, ser. Lecture Notes in Computer Science, E. Ábrahám and J. Havelund, Eds., vol. 8413. Grenoble, France: Springer Berlin Heidelberg, Apr. 2014, pp. 587–592.

[12] D. Baelde, S. Delaune, and L. Hirschi, "A reduced semantics for deciding trace equivalence using constraint systems," in *Proc. 3rd Conference on Principles of Security and Trust (POST'14)*. Springer, 2014, pp. 1–21.

[13] ——, "Partial order reduction for security protocols," in *Proc. 26th International Conference on Concurrency Theory (CONCUR'15)*, ser. LIPIcs, vol. 42. Leibniz-Zentrum für Informatik, 2015, pp. 497–510.

[14] A. L. Blum and M. L. Furst, "Fast planning through planning graph analysis," *Artificial Intelligence*, vol. 90, pp. 281–300, 1997.

[15] H. Kautz and B. Selman, "Planning as satisfiability," in *ECAI*, Vienna, Austria, 1992, pp. 359–363.

[16] A. Armando and L. Compagna, "Sat-based model-checking for security protocols analysis," *International Journal of Information Security*, vol. 7, p. 3–32, 2008.

[17] A. Armando, R. Carbone, and L. Compagna, "SATMC: a SAT-based model checker for security-critical systems," in *Proceedings of the 20th international Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14)*. Springer, 2014, pp. 31–45. [Online]. Available: http://www.ai-lab.it/armando/pub/tacas2014.pdf

[18] S. Delaune and L. Hirschi, "A survey of symbolic methods for establishing equivalence-based properties in cryptographic protocols," *Journal of Logical and Algebraic Methods in Programming*, 2017, to appear, available at https://arxiv.org/abs/1610.08279.

[19] R. Chrétien, V. Cortier, and S. Delaune, "Typing messages for free in security protocols: the case of equivalence properties," in *Proceedings of the 25th International Conference on Concurrency Theory (CONCUR'14)*, ser. Lecture Notes in Computer Science, P. Baldan and D. Gorla, Eds., vol. 8704. Rome, Italy: Springer, Sep. 2014, pp. 372–386. [Online]. Available: http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/CCD-concur14.pdf

[20] https://projects.lsv.ens-cachan.fr/satequiv.

[21] V. Cortier, A. Dallon, and S. Delaune, "SAT-Equiv: an efficient tool for equivalence properties," LSV, ENS Cachan, CNRS, INRIA, Université Paris-Saclay, Cachan (France) ; IRISA, Inria Rennes ; LORIA - Université de Lorraine ; CNRS, Research Report, May 2017. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01529966

[22] M. Abadi and C. Fournet, "Mobile values, new names, and secure communication," in *Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '01. New York, NY, USA: ACM, 2001, pp. 104–115. [Online]. Available: http://doi.acm.org/10.1145/360204.360213

[23] V. Cheval, V. Cortier, and S. Delaune, "Deciding equivalence-based properties using constraint solving," *Theoretical Computer Science*, vol. 492, pp. 1–39, Jun. 2013. [Online]. Available: http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/CCD-tcs13.pdf

[24] J. Clark and J. Jacob, "A survey of authentication protocol literature : Version 1.0," 1997.

[25] B. Blanchet and A. Podelski, "Verification of cryptographic protocols: Tagging enforces termination," in *Foundations of Software Science and Computation Structures (FoSSaCS'03)*.

[26] R. Chrétien, V. Cortier, and S. Delaune, "Checking trace equivalence: How to get rid of nonces?" in *Proceedings of the 20th European Symposium on Research in Computer Security (ESORICS'15)*, ser. Lecture Notes in Computer Science. Vienna, Austria: Springer, 2015.

[27] V. Cortier, A. Dallon, and S. Delaune, "Bounding the number of agents, for equivalence too," in *Proceedings of the 5th International Conference on Principles of Security and Trust (POST'16)*, ser. Lecture Notes in Computer Science, F. Piessens and L. Viganó, Eds. Eindhoven, The Netherlands: Springer, Apr. 2016. [Online]. Available: http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/CDD-post16.pdf

[28] L. Compagna, "SAT-based Model-Checking of Security Protocols," Ph.D. dissertation, Università degli Studi di Genova and the University of Edinburgh (joint programme), September 2005.

[29] N. Een and N. Sörensson, "An Extensible SAT-solver," in *SAT 2003*, 2003, pp. 502–518.

[30] http://www.ntu.edu.sg/home/atiu/spec%2Dprover/.

[31] A. Tiu, "A trace based bisimulation for the spi calculus," in *Programming Languages and Systems*. Springer, 2007, pp. 367–382.

[32] https://projects.lsv.ens-cachan.fr/APTE/author/cheval.

[33] http://akiss.gforge.inria.fr.

[34] R. Chadha, V. Cheval, Ş. Ciobâcă, and S. Kremer, "Automated verification of equivalence properties of cryptographic protocol," *ACM Transactions on Computational Logic*, 2016, to appear. [Online]. Available: https://hal.inria.fr/hal-01306561/document

[35] M. Arapinis, J. Phillips, E. Ritter, and M. D. Ryan, "Statverif: Verification of stateful processes," *Journal of Computer Security*, vol. 22, no. 5, pp. 743–821, 2014. [Online]. Available: http://dx.doi.org/10.3233/JCS-140501