Efficiently deciding equivalence for standard primitives and phases[⋆]

Véronique Cortier[1], Antoine Dallon[1,2,3], and Stéphanie Delaune[3]

[1] LORIA, CNRS, France
[2] LSV, CNRS & ENS Paris-Saclay, France
[3] Univ Rennes, CNRS, IRISA, France

Abstract. Privacy properties like anonymity or untraceability are now well identified, desirable goals of many security protocols. Such properties are typically stated as equivalence properties. However, automatically checking equivalence of protocols often yields efficiency issues.
We propose an efficient algorithm, based on graph planning and SAT-solving. It can decide equivalence for a bounded number of sessions, for protocols with standard cryptographic primitives and phases (often necessary to specify privacy properties), provided protocols are well-typed, that is encrypted messages cannot be confused. The resulting implementation, SAT-Equiv, demonstrates a significant speed-up w.r.t. other existing tools that decide equivalence, covering typically more than 100 sessions. Combined with a previous result, SAT-Equiv can now be used to prove security, for some protocols, for an unbounded number of sessions.

1 Introduction

Security protocols are notoriously difficult to design. A common good practice is to formally analyse protocols using symbolic techniques, in order to spot flaws possibly before their deployment (e.g. TLS 1.3 [20, 4], an avionic protocol [5]). These symbolic techniques are mature for reachability properties like confidentiality or authentication. More recently, this approach has been extended to privacy properties, such as vote secrecy, anonymity, untraceability, or unlinkability. These properties are expressed through equivalences. For example, in the case of biometric passports, an attacker should not be able to distinguish whether she is in contact with Alice's passport or Bob's passport.

Recently, a new tool, SAT-Equiv [16], has been proposed to decide such equivalence properties for security protocols, for a bounded number of sessions. It is based on a standard model-checking approach, namely graph planning [7, 23] and SAT-solving. Intuitively, protocols executions are over-approximated as a

[⋆] The research leading to these results has received funding from the European Research Council under the European Union's horizon 2020 research and innovation program (ERC grant agreement n° 714955-POPSTAR and n° 645865-SPOOC), as well as from the French National Research Agency (ANR) under the project TECAP, and the DGA.

graph planning problem, which allows to consider several possible interleavings in parallel, allowing the analysis of dozen of sessions of a protocol in a few seconds. However, this result is limited to a very small set of primitives, namely symmetric encryption and concatenation.

Our contributions. Building upon this novel approach, we enrich SAT-Equiv in order to cover protocols using asymmetric primitives and/or phases. As for the original SAT-Equiv, we assume a non confusion property: encrypted messages should not be confused, a condition automatically checked by our tool and which can be enforced e.g. through appropriate labelling.

First, we extend SAT-Equiv to cover all standard primitives: symmetric and asymmetric encryption, signatures, and hashes. Since graph planning is a bounded model-checking technique, SAT-Equiv relies on a small model property, that bounds the size of messages. More precisely, [12] guarantees that if there is an attack, then there is a well-typed attack, where messages follow a fix format. This result has been recently extended to standard primitives [14]. The straightforward extension of SAT-Equiv to standard primitives however yields severe efficiency issues. Indeed, unlike the symmetric encryption case, checking whether two sequences of messages are equivalent (i.e. in static equivalence) may require complex tests where the attacker *construct* messages (that is, hash or asymmetrically encrypt messages). We therefore provide a precise characterisation of the set of tests that need to be considered when checking for static equivalence. This characterisation is of independent interest and could be used in other contexts. We also extend SAT-Equiv to consider protocols with phases, which are useful to model game-based properties.

Our extension of SAT-Equiv now provably terminates. In [16], termination can be guaranteed by checking that any state of the planning graph is indeed reachable, which requires to query a SAT-solver at each step. While this provides termination in theory, this yields a non practical algorithm and has not been implemented. Instead, we exhibit a bound on the maximal length of the smallest attack (bounding the attacker steps as well). It is therefore sufficient to stop the construction of the graph planning once this bound has been reached, enforcing termination for free (no computation overhead).

Finally, we have considerably revisited and improved the original implementation of SAT-Equiv. This significant speedup now allows for security proofs for an unbounded number of sessions. Indeed, [13] shows decidability of equivalence, for an unbounded number of sessions, for protocols with an acyclic dependency graph. The notion of dependency graph is introduced in [13] and intuitively captures how the input/output actions of the protocol may use messages from other steps of the protocol. As a corollary, [13] induces a bound on the number of sessions that needs to be considered for an attack, which depends on the size and structure of the graph. This bound can be rather large (50 to 100 sessions, even on small examples) but SAT-Equiv is now able to reach such bounds.

These novelties are implemented in an extension of SAT-Equiv and compared with the other tools of the literature, namely Spec [25], Akiss [8] and the very recent DeepSec [11] tool. Our experiments show that SAT-Equiv is much faster on all the examples, allowing to reach typically more than 100 sessions. As an application, we consider two protocols, Denning-Sacco and Needham-Schroeder symmetric keys, shown to have acyclic dependency graphs in [13]. Considering the necessary number of sessions as induced by [13], we establish trace equivalence for these two protocols, for an unbounded number of sessions.

Due to lack of space, the reader is referred to the companion technical report [17] for the missing proofs and additional details.

Related work. There are two main families of tools to analyse equivalence properties on security protocols. Some tools prove equivalence for an arbitrary number of sessions, that is, no matter how often a protocol is used. The main tools in this category are ProVerif [6], Tamarin [24], Maude-NPA [22], Type-Eq [18]. Maude-NPA often suffers from termination issues when used for equivalence properties. Type-Eq [18, 19] is a sound (but incomplete) type-checker for equivalence properties that has good performance. It requires that protocols have a similar structure. ProVerif and Tamarin work well in practice. They actually prove a stronger notion of equivalence, diff-equivalence, that also requires that the two considered protocols have a very similar structure. Moreover, equivalence properties are undecidable in general for an unbounded number of sessions. Therefore, ProVerif may not terminate and Tamarin may need some user guidance.

A second approach consists in *deciding* equivalence, for a bounded number of sessions. Spec [25] is one of the first tool that decides equivalence of security protocols but it does not scale well when the number of sessions grows (it can typically handle up to three sessions for small protocols). DeepSec [11] is a very recent tool that builds upon Akiss [8] and Apte [9]. All these tools analyse symbolic executions and typically have to consider all possible interleavings between the roles of the protocol, which often raises efficiency issues.

2 Model

Protocols are modeled through a process algebra, in the spirit of the applied-pi calculus [1]. We consider here a model similar to the ones used *e.g.* in [16, 14].

2.1 Term algebra

As usual, messages are modeled by terms. Private data are represented through an infinite set $\mathcal{N}$ of *names* used to model *e.g.* keys or nonces. We consider an infinite set $\mathcal{C}_0$ of constants to represent public data such as agent names or attacker's nonces or keys. We consider also two sets of *variables* $\mathcal{X}$ and $\mathcal{W}$. Variables in $\mathcal{X}$ model arbitrary data expected by the protocol, while variables in $\mathcal{W}$ are used to store messages learnt by the attacker. A *data* is either a constant, a variable, or a name. Cryptograhic primitives are represented by function symbols. We consider the *signature* Σ parameterised by $n \geq 2$:

– $\Sigma_{\mathsf{c}} = \{\mathsf{senc}, \mathsf{aenc}, \mathsf{hash}, \mathsf{pub}, \mathsf{sign}, \mathsf{vk}, \mathsf{ok}\} \cup \{\langle\ \rangle_k \mid 2 \leq k \leq n\}$;

- $\Sigma_{\mathsf{d}} = \{\mathsf{sdec},\ \mathsf{adec},\ \mathsf{getmsg}\} \cup \{\mathsf{proj}_j^k \mid 2 \leq k \leq n \text{ and } 1 \leq j \leq k\}$; and
- $\Sigma = \Sigma_{\mathsf{c}} \cup \Sigma_{\mathsf{d}} \cup \{\mathsf{check}\}$.

The symbols senc, aenc, sdec, and adec of arity 2 are used to model resp. symmetric and asymmetric encryption. We also consider signature sign and hash function hash. Concatenation of messages is modeled through tuple operators together with their projection functions. For example, $\langle m_1, m_2, m_3\rangle_3$ represents the concatenation of the three messages m_1, m_2, and m_3. It is syntactically different from the nested pairs $\langle m_1, \langle m_2, m_3\rangle_2\rangle_2$. These two representations correspond to different implementation choices. We distinguish between *constructors* in Σ_{c} and *destructors* in Σ_{d}. The symbol check of arity 2, which corresponds to the verification of a signature, is neither a destructor nor a constructor. The set of terms built from a signature $\mathcal{F}$ and a set of data D is denoted $\mathcal{T}(\Sigma, D)$. Given a term u, we denote $St(u)$ the set of its *subterms*, $vars(u)$ the set of its *variables*, and $\mathsf{root}(u)$ its root symbol. A term is *ground* if it contains no variable. The application of a substitution σ to a term u is written $u\sigma$. We denote $dom(\sigma)$ its *domain* and $img(\sigma)$ its *image*. Two terms u_1 and u_2 are *unifiable* when there exists a substitution σ such that $u_1\sigma = u_2\sigma$.

We consider two *sorts*: atom and bitstring. The sort atom represents atomic data like nonces or keys while bitstring models arbitrary messages. Names in $\mathcal{N}$ and constants in $\mathcal{C}_0$ have sort atom. Any $\mathsf{f} \in \Sigma_{\mathsf{c}}$ comes with its sorted arity:

$$\begin{array}{ll}
\langle\ \rangle_k : \mathsf{bitstring} \times \cdots \times \mathsf{bitstring} \rightarrow \mathsf{bitstring} & \mathsf{ok} : \quad\quad \rightarrow \mathsf{bitstring} \\
\mathsf{senc} : \mathsf{bitstring} \times \mathsf{atom} \rightarrow \mathsf{bitstring} & \mathsf{pub} : \ \mathsf{atom} \rightarrow \mathsf{bitstring} \\
\mathsf{aenc} : \mathsf{bitstring} \times \mathsf{bitstring} \rightarrow \mathsf{bitstring} & \mathsf{vk} : \ \mathsf{atom} \rightarrow \mathsf{bitstring} \\
\mathsf{sign} : \mathsf{bitstring} \times \mathsf{atom} \rightarrow \mathsf{bitstring} & \mathsf{hash} : \mathsf{bitstring} \rightarrow \mathsf{bitstring}
\end{array}$$

Given $D \subseteq \mathcal{C}_0 \uplus \mathcal{X}$, the set $\mathcal{T}_0(\Sigma_{\mathsf{c}}, D)$ is the set of terms t in $\mathcal{T}(\Sigma_{\mathsf{c}}, D)$ such that *(i)* for any term $\mathsf{pub}(u)$ (resp. $\mathsf{vk}(u)$) in $St(t)$, u is of sort atom; *(ii)* for any $\mathsf{aenc}(u, v) \in St(t)$, $v = \mathsf{pub}(v')$ for some v'. Terms in $\mathcal{T}_0(\Sigma_{\mathsf{c}}, \mathcal{N} \uplus \mathcal{C}_0)$ are called *messages*. Intuitively, messages are terms with atomic keys.

The properties of the cryptographic primitives are reflected through the following convergent rewriting rules.

$$\begin{array}{ll}
\mathsf{sdec}(\mathsf{senc}(x, y), y) \rightarrow x & \mathsf{adec}(\mathsf{aenc}(x, \mathsf{pub}(y)), y) \rightarrow x \\
\mathsf{getmsg}(\mathsf{sign}(x, y)) \rightarrow x & \mathsf{check}(\mathsf{sign}(x, y), \mathsf{vk}(y)) \rightarrow \mathsf{ok} \\
\mathsf{proj}_j^k(\langle x_1, \ldots, x_k\rangle_k) \rightarrow x_j & \text{with } 2 \leq k \leq n \text{ and } 1 \leq j \leq k
\end{array}$$

A term u can be rewritten into v if there is a position p in u, and a rewriting rule $\mathsf{g}(t_1, ..., t_n) \rightarrow t$ such that $u|_p = \mathsf{g}(t_1, \ldots, t_n)\theta$ for some substitution θ, and $v = u[t\theta]_p$, i.e. u in which the subterm at position p has been replaced by $t\theta$. Moreover, we assume that $t_1\theta, \ldots, t_n\theta$ as well as $t\theta$ are messages, in particular they do not contain destructor symbols. As usual, we denote $\rightarrow^*$ the reflexive-transitive closure of $\rightarrow$, and $u{\downarrow}$ the *normal form* of a term u.

An attacker builds her own messages by applying public function symbols to terms she already knows and which are available through variables in $\mathcal{W}$. Formally, a computation done by the attacker is a *recipe*, *i.e.* a term in $\mathcal{T}(\Sigma, \mathcal{W} \uplus \mathcal{C}_0)$.

2.2 Process algebra

We consider processes that may receive and send messages. We assume that each process communicates on a dedicated public channel. In practice, IP addresses and sessions identifiers are typically used to desambiguate which message is addressed to whom and for which session. Of course, these channels may be freely manipulated by the attacker. Since we consider equivalence properties, distinct (public) channels provide more abilities for the adversary to distinguish between protocols. Formally, given a set $\mathcal{C}h$ of channels, we consider the fragment of simple processes without replication built on basic processes as defined *e.g.* in [10].

Definition 1. *A* basic processes *is defined as follows:*

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

with $u_1, u_2 \in \mathcal{T}_0(\Sigma_{\mathsf{c}}, \mathcal{C}_0 \uplus \mathcal{N} \uplus \mathcal{X})$*,* $c \in \mathcal{C}h$*, and increasing phase numbers. A* simple process *is a multiset of basic processes on pairwise distinct channels. A* protocol *is a simple process such that all its variables are in the scope of an input.*

The process 0 does nothing and we often omit it. 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 $m = u_1\sigma$. Note that checking whether a received message has the expected form is done through pattern-matching instead of explicit tests. The process "$\mathsf{out}(c, u_2).P$" emits u_2 on c, and then behaves like P. Our calculus also has a phase instruction, in the spirit of [6], denoted $i{:}P$. This instruction is useful to model security requirements, for example in case the attacker interacts with the protocol before being given some secret.

Example 1. As an illustrative example, we consider a simplified version of the Denning-Sacco protocol which is a key distribution protocol relying on asymmetric encryption and signature. Informally, the protocol is as follows.

$$A \rightarrow B : \mathsf{aenc}(\mathsf{sign}(\langle A, B, K_{ab}\rangle, \mathsf{prv}(A)), \mathsf{pub}(B))$$

The agents A and B aim at authenticating each other and establishing a fresh session key K_{ab}. We model this protocol in our formalism through the simple process $\mathcal{P}_{\mathsf{DS}} = \{P_A; P_B\}$ where $P_A = \mathsf{out}(c_A, \mathsf{aenc}(\mathsf{sign}(\langle \mathsf{a}, \mathsf{b}, k_{ab}\rangle_3, sk_a), \mathsf{pub}(sk_b))).0$ and $P_B = \mathsf{in}(c_B, \mathsf{aenc}(\mathsf{sign}(\langle \mathsf{a}, \mathsf{b}, x\rangle_3, sk_a), \mathsf{pub}(sk_b))).0$ where sk_a, sk_b, and k_{ab} are names, a and b are constants, and x is a variable.

The operational semantics of a process is defined using a relation over configurations. A *configuration* is a tuple $(\mathcal{P}; \phi; \sigma; i)$ with $i \in \mathbb{N}$ and such that:

- $\mathcal{P}$ is a multiset of processes (not necessarily ground);
- $\phi = \{\mathsf{w}_1 \rhd m_1, \ldots, \mathsf{w}_n \rhd m_n\}$ is a *frame*, *i.e.* a substitution where $\mathsf{w}_1, \ldots, \mathsf{w}_n$ are variables in $\mathcal{W}$, and $m_1, \ldots, m_n$ are messages;
- σ is a substitution such that $fv(\mathcal{P}) \subseteq dom(\sigma)$, and $img(\sigma)$ are messages.

A configuration is said to be *initial* when $\sigma = \emptyset$. Intuitively, $\mathcal{P}$ represents the processes that still remain to be executed; ϕ represents the sequence of messages

In $\quad (i{:}\mathsf{in}(c,u).P \cup \mathcal{P};\phi;\sigma;i) \xrightarrow{\mathsf{in}(c,R)} (i{:}P \cup \mathcal{P};\phi;\sigma \uplus \sigma_0;i)$ where R is a recipe such that $R\phi\downarrow$ is a message, and $R\phi\downarrow = (u\sigma)\sigma_0$ for σ_0 with $dom(\sigma_0) = vars(u\sigma)$.

Out $\quad (i{:}\mathsf{out}(c,u).P \cup \mathcal{P};\phi;\sigma;i) \xrightarrow{\mathsf{out}(c,\mathsf{w})} (i{:}P \cup \mathcal{P};\phi \cup \{\mathsf{w} \triangleright u\sigma\};\sigma;i)$
with w a fresh variable from $\mathcal{W}$, and $u\sigma$ is a message.

Move $\quad (\mathcal{P};\phi;\sigma;i) \xrightarrow{\mathsf{phase}\ i'} (\mathcal{P};\phi;\sigma;i')$ with $i' > i$.

Phase $\quad (i'{:}i''{:}P \cup \mathcal{P};\phi;\sigma;i) \xrightarrow{\tau} (i''{:}P \cup \mathcal{P};\phi;\sigma;i)$

Fig. 1. Semantics for processes

that have been learnt so far by the attacker, and σ stores the value of the variables that have already been instantiated. We write P instead of $0{:}P$ and $P \uplus \mathcal{P}$ instead of $\{P\} \uplus \mathcal{P}$. Given a protocol $\mathcal{P}$, we also often write $\mathcal{P}$ instead of $(\mathcal{P};\emptyset;\emptyset;0)$. The operational semantics is induced by the relation $\xrightarrow{\alpha}$ over configurations defined in Figure 1. For example, the In rule defines how messages can be input on a (public) channel: the adversary may send any message, provided she can construct it through a recipe R applied on her previous knowledge ϕ. Note that only messages can be received (and sent). The relation $\xrightarrow{\mathsf{tr}}$ between configurations (where tr is a possibly empty sequence of actions) is defined in the usual way. Given a configuration $\mathcal{K}$, we write:

$$\mathsf{trace}(\mathcal{K}) = \{(\mathsf{tr},\phi) \mid \mathcal{K} \xrightarrow{\mathsf{tr}} (\mathcal{P}';\phi;\sigma;i) \text{ for some configuration } (\mathcal{P}';\phi;\sigma;i)\}.$$

Example 2. Continuing Example 1, let $\mathcal{K}_{\mathsf{DS}} = (\{P_A;P_B;P_{B'}\};\phi_0;\emptyset;0)$ where $P_{B'}$ models an additional session of the role B obtained by simply renaming c_B and x with c'_B and x'. The frame $\phi_0 = \{\mathsf{w}_a \triangleright \mathsf{vk}(sk_a), \mathsf{w}_b \triangleright \mathsf{pub}(sk_b)\}$ models the fact that the attacker initially knows the public key of b and the verification key of a. We consider a simple scenario without dishonest participant. The trace $\mathsf{tr}_0 = \mathsf{out}(c_A,\mathsf{w}_1).\mathsf{in}(c_B,\mathsf{w}_1).\mathsf{in}(c'_B,\mathsf{w}_1)$ is executable from $\mathcal{K}_{\mathsf{DS}}$, and yields $\phi = \phi_0 \uplus \{\mathsf{w}_1 \triangleright \mathsf{aenc}(\mathsf{sign}(\langle \mathsf{a},\mathsf{b},k_{ab}\rangle_3, sk_a), \mathsf{pub}(sk_b))\}$, i.e. $(\mathsf{tr}_0,\phi) \in \mathsf{trace}(\mathcal{K}_{\mathsf{DS}})$.

2.3 Type-compliance

We present here our main assumption on protocols. Intuitively, we assume that ciphertexts cannot be confused, and we rely for this on a notion of typing system.

Definition 2. *A* typing system *is a pair* $(\mathcal{T}_{\mathsf{init}},\delta)$ *where* $\mathcal{T}_{\mathsf{init}}$ *is a set of elements called* initial types*, and* δ *is a function mapping data in* $\mathcal{C}_0 \uplus \mathcal{N} \uplus \mathcal{X}$ *to types* τ*:*

$$\tau,\tau_1,\tau_2 = \tau_0 \mid \mathsf{f}(\tau_1,\ldots,\tau_n) \text{ with } \mathsf{f} \in \Sigma_{\mathsf{c}} \text{ and } \tau_0 \in \mathcal{T}_{\mathsf{init}}$$

Then, δ *is extended to constructor terms as follows:*

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

A configuration is type-compliant if two unifiable encrypted subterms have the same type. We write $ESt(t)$ for the set of *encrypted subterms* of t, i.e. $ESt(t) = \{u \in St(t) \mid u \text{ is of the form } \mathsf{f}(u_1,\ldots,u_n) \text{ and } \mathsf{f} \neq \langle\ \rangle_i\}$.

Definition 3. *An initial configuration $\mathcal{K}$ is type-compliant w.r.t. a typing system $(\mathcal{T}_{\mathsf{init}}, \delta)$ if for every $t, t' \in ESt(\mathcal{K})$ we have that t and t' unifiable implies that $\delta(t) = \delta(t')$.*

Example 3. Continuing our running example, we consider the typing system generated from $\mathcal{T}_{\mathsf{DS}} = \{\tau_{\mathsf{a}}, \tau_{\mathsf{b}}, \tau_k, \tau_{sk}\}$ of initial types, and the function δ_{DS} that associates the expected type to each constant/name ($\delta_{\mathsf{DS}}(\mathsf{a}) = \tau_{\mathsf{a}}$, $\delta_{\mathsf{DS}}(k_{ab}) = \tau_k$, etc), and such that $\delta_{\mathsf{DS}}(x) = \delta_{\mathsf{DS}}(x') = \tau_k$. We have that $\mathcal{K}_{\mathsf{DS}}$ is type-compliant w.r.t. $(\mathcal{T}_{\mathsf{DS}}, \delta_{\mathsf{DS}})$: unifiable encrypted subterms occurring in the configuration have the same type since $\delta_{\mathsf{DS}}(x) = \delta_{\mathsf{DS}}(x') = \delta_{\mathsf{DS}}(k_{ab})$.

Type-compliant protocols have the property that, when looking for attacks, it is sufficient to consider well-typed execution: $\mathcal{K} \xrightarrow{\mathsf{tr}} (\mathcal{P}; \phi; \sigma; i)$ is *well-typed* w.r.t. a typing system $(\mathcal{T}_{\mathsf{init}}, \delta)$, if σ is a well-typed substitution, i.e. every variable of its domain has the same type as its image.

2.4 Trace equivalence

Many privacy properties such as vote-privacy or untraceability are expressed as trace equivalence [21, 2]. Intuitively, two configurations are trace equivalent if an attacker cannot tell with which of the two configurations she is interacting. We first introduce a notion of equivalence (actually, inclusion) between frames.

Definition 4. *Two frames ϕ_1 and ϕ_2 are in* static inclusion*, written $\phi_1 \sqsubseteq_s \phi_2$, when $dom(\phi_1) = dom(\phi_2)$, and:*

- *for any recipe R, we have that $R\phi_1\!\downarrow$ is a message implies that $R\phi_2\!\downarrow$ is a message;*
- *for any recipes R, R' such that $R\phi_1\!\downarrow$, $R'\phi_1\!\downarrow$ are messages, we have that: $R\phi_1\!\downarrow = R'\phi_1\!\downarrow$ implies $R\phi_2\!\downarrow = R'\phi_2\!\downarrow$.*

Intuitively, ϕ_1 is included in ϕ_2 if any recipe producing a message in ϕ_1 also produces a message in ϕ_2 and if any equality satisfied in ϕ_1 is also satisfied in ϕ_2.

Example 4. We consider $\phi_1 = \phi \uplus \{\mathsf{w}_2 \triangleright \mathsf{senc}(\mathsf{m}_1, k_{ab}), \mathsf{w}'_2 \triangleright \mathsf{senc}(\mathsf{m}_1, k_{ab})\}$, and $\phi_2 = \phi \uplus \{\mathsf{w}_2 \triangleright \mathsf{senc}(\mathsf{m}_2, k), \mathsf{w}'_2 \triangleright \mathsf{senc}(\mathsf{m}_2, k')\}$ where $\mathsf{m}_1, \mathsf{m}_2 \in \mathcal{C}_0$. We have that $\mathsf{w}_2\phi_1\!\downarrow = \mathsf{w}'_2\phi_1\!\downarrow$ whereas this equality does not hold in ϕ_2. Hence $\phi_1 \not\sqsubseteq_s \phi_2$.

Trace inclusion is the active counterpart of static inclusion. Two configurations are in trace inclusion if, however the attacker behaves, the resulting sequences of messages observed by the attacker are in static inclusion.

Definition 5. *Let $\mathcal{K}$ and $\mathcal{K}'$ be two configurations. We have that $\mathcal{K} \sqsubseteq_t \mathcal{K}'$, if for every $(\mathsf{tr}, \phi) \in \mathsf{trace}(\mathcal{K})$, there exists $(\mathsf{tr}, \phi') \in \mathsf{trace}(\mathcal{K}')$ such that $\phi \sqsubseteq_s \phi'$.*

We easily derive a notion of trace equivalence: two configurations $\mathcal{K}$ and $\mathcal{K}'$ are trace equivalence, denoted $\mathcal{K} \approx_t \mathcal{K}'$, if $\mathcal{K} \sqsubseteq_t \mathcal{K}'$ and $\mathcal{K}' \sqsubseteq_t \mathcal{K}$. This notion of trace equivalence slightly differs from the one used in e.g. [12] but they actually coincide on the class of protocols we consider in this paper [8].

Example 5. To model secrecy of the key k_{ab}, we define *strong secrecy* of k_{ab} by requiring that k_{ab} is indistinguishable from a fresh value. Formally, we consider P_B^1 (resp. $P_{B'}^1$) obtained by replacing the process 0 with $1{:}\mathsf{out}(c_B, \mathsf{senc}(\mathsf{m}_1, x))$ (resp. $1{:}\mathsf{out}(c'_B, \mathsf{senc}(\mathsf{m}_1, x'))$). On the other side of the equivalence, we consider P_B^2 and $P_{B'}^2$ obtained by replacing the process 0 with $1{:}\mathsf{out}(c_B, \mathsf{senc}(\mathsf{m}_2, k))$ (resp. $1{:}\mathsf{out}(c'_B, \mathsf{senc}(\mathsf{m}_2, k'))$) with fresh names k and k'.

$$\mathcal{K}^1_{\mathsf{DS}} = (\{P_A; P_B^1; P_{B'}^1\}; \phi_0) \text{ and } \mathcal{K}^2_{\mathsf{DS}} = (\{P_A; P_B^2; P_{B'}^2\}; \phi_0).$$

Then, we can show that $\mathcal{K}^1_{\mathsf{DS}} \not\sqsubseteq_t \mathcal{K}^2_{\mathsf{DS}}$ since k_{ab} is not strongly secret. An attacker can replay the message sent by A due to lack of freshness. This is exemplified by the trace $\mathsf{tr}_0.\mathsf{out}(c_B, \mathsf{w}_2).\mathsf{out}(c'_B, \mathsf{w}'_2)$ and the test given in Example 4.

3 From static inclusion to planning

The overall objective of this paper is to provide a practical algorithm for deciding trace inclusion (and thus trace equivalence) relying on graph planning and SAT solving. We start here by explaining how to build a planning problem from two frames such that the planning problem has a solution if, and only if, the two corresponding frames are not in static inclusion.

3.1 Planning problems

We first recall the definition of a planning problem, slightly simplified from [15]. 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 6. *A* planning system *is tuple $\langle \mathcal{F}act, \mathcal{I}nit, \mathcal{R}ule \rangle$ where $\mathcal{F}act$ is a set of ground 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* $\mathsf{Pre} \to \mathsf{Add}; \mathsf{Del}$ *where* Pre*,* Add*,* Del *are finite sets of facts such that* $\mathsf{Add} \cap \mathsf{Del} = \emptyset$*,* $\mathsf{Del} \subseteq \mathsf{Pre}$*. We write* $\mathsf{Pre} \to \mathsf{Add}$ *when* $\mathsf{Del} = \emptyset$*.*

Given a rule $r \in \mathcal{R}ule$ of the form $\mathsf{Pre} \to \mathsf{Add}; \mathsf{Del}$, we denote $\mathsf{Pre}(r) = \mathsf{Pre}$, $\mathsf{Add}(r) = \mathsf{Add}$, and $\mathsf{Del}(r) = \mathsf{Del}$. If $S \subseteq \mathcal{F}act$ are such that $\mathsf{Pre}(r) \subseteq S$, then we say that the rule is *applicable* in S, denoted $S \xrightarrow{r} S'$, and the state $S' = (S \smallsetminus \mathsf{Del}) \cup \mathsf{Add}$ is the state resulting from the application of r to S. We allow some rules to be applied in parallel when no facts are deleted. Given $S \subseteq \mathcal{F}act$, and a set of rules $\{r_1, \ldots, r_k\}$ such that $\mathsf{Del}(r_i) = \emptyset$ and $\mathsf{Pre}(r_i) \subseteq S$ for any $i \in \{1, \ldots, k\}$, $\{r_1, \ldots, r_k\}$ is *applicable* in S, denoted $S \xrightarrow{\{r_1,\ldots,r_k\}} S'$, and the state $S' = \bigcup_{i=1}^k \mathsf{Add}(r_i) \cup S$ is the state resulting from the application of $\{r_1, \ldots, r_k\}$ to S.

A *planning path* from $S_0 \subseteq \mathcal{F}act$ to $S_n \subseteq \mathcal{F}act$ is a sequence $r_1, \ldots, r_n$ made of rules or sets of rules in $\mathcal{R}ule$ such that $S_0 \xrightarrow{r_1} S_1 \xrightarrow{r_2} \ldots S_{n-1} \xrightarrow{r_n} S_n$ for some states $S_1, \ldots, S_{n-1} \subseteq \mathcal{F}act$. 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 $S_f \subseteq \mathcal{F}$ 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 $S_f \subseteq S_n$.

A transition $S \xrightarrow{\{r_1,\dots,r_k\}} S'$ can be mimicked by $S \xrightarrow{r_1} S_1 \xrightarrow{r_2} \dots \xrightarrow{r_k} S'$, thus the possibility of applying set of rules in a single step does not change the set of reachable states from a given state S. However, this allows us to consider plans of smaller length and will be useful later on to derive a tight bound and ensure the termination of our algorithm.

In this section, we explain the translation of static inclusion into a planning problem. We consider an (infinite) set $\mathcal{F}act_0$ of facts that represent the attacker's knowledge, i.e. formulas of the form $\mathsf{att}(u_P, u_Q)$ where u_P and u_Q are messages, plus a special symbol bad. Intuitively, $\mathsf{att}(u_P, u_Q)$ means that the attacker knows u_P in the "left" frame, while she knows u_Q in the "right" one.

3.2 Attacker analysis rules

Following [16], we first describe the planning rules that correspond to the analysis part of the attacker behaviours. We start by describing a set of abstract rules $\mathsf{R_{Ana}}$ that will be instantiated later on, yielding a (concrete) planning system.

$$\begin{aligned}
\mathsf{att}(\langle x_1,\dots,x_k\rangle_k, \langle y_1,\dots,y_k\rangle_k) &\to \mathsf{att}(x_i, y_i) \text{ with } i \le k\\
\mathsf{att}(\mathsf{senc}(x_1,x_2), \mathsf{senc}(y_1,y_2)),\ \mathsf{att}(x_2,y_2) &\to \mathsf{att}(x_1,y_1)\\
\mathsf{att}(\mathsf{aenc}(x_1,\mathsf{pub}(x_2)), \mathsf{aenc}(y_1,\mathsf{pub}(y_2))),\ \mathsf{att}(x_2,y_2) &\to \mathsf{att}(x_1,y_1)\\
\mathsf{att}(\mathsf{sign}(x_1,x_2), \mathsf{sign}(y_1,y_2)) &\to \mathsf{att}(x_1,y_1)
\end{aligned}$$

These rules correspond to the attacker's ability to project, decrypt, and retrieve messages from their signature. There is no Del since the attacker never forgets. Given a rule $r \in \mathsf{R_{Ana}}$, we explain how to compute its concretization denoted $\mathsf{Concrete}(r)$. Formally, we have that $\mathsf{Concrete}(r) = \mathsf{Concrete}^+(r) \cup \mathsf{Concrete}^-(r)$.

$\mathsf{Concrete}^+(r)$. The positive concretizations of r consist of instantiating r such that the resulting terms are messages. More formally, we have:

$$\mathsf{Concrete}^+(r) = \{r\sigma \mid \sigma \text{ substitution such that } r\sigma \text{ only involve messages.}\}$$

$\mathsf{Concrete}^-(r)$. We say that a sequence of ground facts $\mathsf{att}(u_1, v_1), \dots, \mathsf{att}(u_k, v_k)$ left-unifies with a sequence $\mathsf{att}(u'_1, v'_1), \dots, \mathsf{att}(u'_k, v'_k)$ if there exists σ such that $u'_1\sigma = u_1, \dots, u'_k\sigma = u_k$ (and symmetrically for right-unification). Given an abstract attacker rule $r = \mathsf{Pre} \to \mathsf{Add}$, we define $\mathsf{Concrete}^-(r)$ as the set containing $f_1, \dots, f_k \to \mathsf{bad}$ for any sequence of facts $f_1, \dots, f_k \in \mathcal{F}act_0$ such that $f_1, \dots, f_k$ left-unifies with Pre, whereas $f_1, \dots, f_k$ does not right-unify with Pre.

Example 6. The negative concretizations of the abstract rule corresponding to asymmetric decryption are all the concrete rules of the form

$$\mathsf{att}(\mathsf{aenc}(u_1, \mathsf{pub}(u_2)), v),\ \mathsf{att}(u_2, v') \to \mathsf{bad}$$

where u_1, u_2, v, v', $\mathsf{aenc}(u_1, \mathsf{pub}(u_2))$ are messages, whereas $\mathsf{adec}(v, v')\downarrow$ is not.

3.3 Static inclusion

According to Definition 4, to break static inclusion, an attacker may build new terms (using both analysis and synthesis rules) but also check for equalities and computation failures. To encode static inclusion using planning in an efficient way, we need to strictly control the terms that an attacker has to synthetise.

We say that R is *destructor-only* if $R \in \mathcal{T}(\Sigma_{\mathsf{d}}, \mathcal{C}_0 \cup \mathcal{W})$. It is *simple* if there exists destructor-only recipes $R_1, \ldots, R_k$, and a context C made of constructors such that $R = C[R_1, \ldots, R_k]$.

Definition 7. *Let ϕ, ψ be such that $dom(\phi) = dom(\psi)$. We write $\phi \sqsubseteq_s^{\mathsf{simple}} \psi$ if:*

1. *For each destructor-only recipe R such that $R\phi\downarrow$ is a (resp. atomic) message, $R\psi\downarrow$ is a (resp. atomic) message.*
2. *For each simple recipe R and destructor-only recipe R' such that $R\phi\downarrow, R'\phi\downarrow$ are messages and $R\phi\downarrow = R'\phi\downarrow$, we have that $R\psi\downarrow = R'\psi\downarrow$.*
3. *For each destructor-only recipes R, R', if $R\phi\downarrow = \mathsf{sign}(t, s)$, and $R'\phi\downarrow = \mathsf{vk}(s)$ for some term t and atom s, then $R\psi\downarrow = \mathsf{sign}(t', s')$, and $R'\psi\downarrow = \mathsf{vk}(s')$ for some term t' and atom s'.*
4. *For each destructor-only recipe R, such that $R\phi\downarrow = \mathsf{pub}(s)$ for some atom s, $R\psi\downarrow = \mathsf{pub}(s')$ for some atom s'.*

We write $\phi \sqsubseteq_s^{\mathsf{simple}^+} \psi$ when the test described at item 2 *is only performed when (i) either R is destructor-only; (ii) or* $\mathsf{root}(R) \notin \{\mathsf{senc}\} \cup \{\langle\ \rangle_k \mid 2 \leq k \leq n\}$, *and* $\mathsf{root}(R') \neq \mathsf{adec}$.

This notion of static inclusion is equivalent to the original one.

Lemma 1. *Let ϕ and ψ be two frames having the same domain. We have that:*

$$\phi \sqsubseteq_s \psi \;\Leftrightarrow\; \phi \sqsubseteq_s^{\mathsf{simple}} \psi \;\Leftrightarrow\; \phi \sqsubseteq_s^{\mathsf{simple}^+} \psi.$$

From this new characterisation of static inclusion $\sqsubseteq_s^{\mathsf{simple}}$, we derive the planning rules that capture all the cases of failures with those in $\mathsf{Concrete}^-(\mathsf{R}_{\mathsf{Ana}})$.

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

$$\mathcal{R}_{\mathsf{fail}}^{\mathsf{pub}} = \{\mathsf{att}(\mathsf{pub}(u), v) \to \mathsf{bad} \mid v \text{ is not of the form } \mathsf{pub}(v')\}$$

$$\mathcal{R}_{\mathsf{fail}}^{\mathsf{check}} = \left\{ \begin{array}{l} \mathsf{att}(\mathsf{sign}(u_1, u_2), v_1) \\ \mathsf{att}(\mathsf{vk}(u_2), v_2) \end{array} \to \mathsf{bad} \mid \mathsf{check}(v_1, v_2)\downarrow \text{ is not a message} \right\}$$

$$\mathcal{R}_{\mathsf{fail}}^{\mathsf{test}} = \left\{ \begin{array}{l} \mathsf{att}(u_1, v_1), \ldots, \mathsf{att}(u_k, v_k) \\ \mathsf{att}(C[u_1, \ldots, u_k], v) \end{array} \to \mathsf{bad} \mid \begin{array}{l} C \text{ is a constructor context,} \\ C[u_1, \ldots, u_k] \in St(\phi) \cup \mathcal{C}_0 \\ v \neq C[v_1, \ldots, v_k]. \end{array} \right\}$$

Actually, not all subterms of $St(\phi)$ need to be considered. Therefore, we consider an optimised version that captures only the terms that may not be reconstructed from their subterms. Formally, $St_{\mathsf{opti}}(t)$ is defined as follows.

- $St_{\mathsf{opti}}(\langle t_1, t_2\rangle) = St_{\mathsf{opti}}(t_1) \cup St_{\mathsf{opti}}(t_2)$;
- $St_{\mathsf{opti}}(\mathsf{senc}(t_1, t_2)) = St_{\mathsf{opti}}(t_1)$;

– $St_{\mathsf{opti}}(\mathsf{aenc}(t_1, t_2)) = \{\mathsf{aenc}(t_1, t_2)\} \cup (St_{\mathsf{opti}}(t_1) \smallsetminus \{t_1\})$
– $St_{\mathsf{opti}}(\mathsf{sign}(t_1, t_2)) = \{\mathsf{sign}(t_1, t_2)\} \cup St_{\mathsf{opti}}(t_1)$;
– $St_{\mathsf{opti}}(\mathsf{f}(t)) = \{\mathsf{f}(t)\}$ with $\mathsf{f} \in \{\mathsf{hash}, \mathsf{pub}, \mathsf{vk}\}$.

Thanks to the fact that $\sqsubseteq_s^{\mathsf{simple}^+}$ is equivalent to static inclusion, we may only consider simple recipes which evaluation yields a term in $St_{\mathsf{opti}}(\phi)$.

Lemma 2. *Let ϕ be a frame, $R = C[R_1, \dots, R_k]$ be a simple recipe such that $\mathsf{root}(R) \notin \{\mathsf{senc}\} \cup \{\langle\ \rangle_k \mid 2 \leq k \leq n\}$, and R' be a destructor-only recipe such that $\mathsf{root}(R') \neq \mathsf{adec}$. Assume that $R\phi\!\downarrow$ and $R'\phi\!\downarrow$ are both messages such that $R\phi\!\downarrow = R'\phi\!\downarrow$. We have that either C is the empty context, or $R\phi\!\downarrow \in St_{\mathsf{opti}}(\phi) \cup \mathcal{C}_0$.*

Therefore, $\mathcal{R}_{\mathsf{fail}}^{\mathsf{test}}$ can be replaced by the following (smaller) set of rules:

$$\mathcal{R}_{\mathsf{fail}}^{\mathsf{test}_1} = \{\mathsf{att}(u_1, v_1), \mathsf{att}(u_1, v_2) \to \mathsf{bad} \mid v_1 \neq v_2\}$$

$$\mathcal{R}_{\mathsf{fail}}^{\mathsf{test}_2} = \{\mathsf{att}(u_1, v_1), \dots, \mathsf{att}(u_k, v_k), \mathsf{att}(C[u_1, \dots, u_k], v) \to \mathsf{bad} \mid C \text{ is a non-empty constructor context, } C[u_1, \dots, u_k] \in St_{\mathsf{opti}}(\phi) \cup \mathcal{C}_0, \text{ and } v \neq C[v_1, \dots, v_k].\}$$

Let ϕ and ψ be two frames with $dom(\phi) = dom(\psi)$ and built using constants from $\mathcal{C} \subseteq \mathcal{C}_0$. The set of facts associated to ϕ and ψ is defined as follows:

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

Two frames are in static inclusion if, and only if, the corresponding planning system has no solution. Actually, when the frames are not in static inclusion, we provide a bound on the length of the (minimal) plan witnessing this fact.

Proposition 1. *Let ϕ and ψ be two frames with $dom(\phi) = dom(\psi)$, and $\Theta = \langle \mathcal{F}act_0, \mathsf{Fact}_{\mathcal{C}_0}(\phi, \psi), \mathcal{R} \rangle$ where*

$$\mathcal{R} = \mathsf{Concrete}(\mathsf{R}_{\mathsf{Ana}}) \cup \mathcal{R}_{\mathsf{fail}}^{\mathsf{test}_1} \cup \mathcal{R}_{\mathsf{fail}}^{\mathsf{test}_2} \cup \mathcal{R}_{\mathsf{fail}}^{\mathsf{atom}} \cup \mathcal{R}_{\mathsf{fail}}^{\mathsf{check}} \cup \mathcal{R}_{\mathsf{fail}}^{\mathsf{pub}}.$$

Let $\Pi = \langle \Theta, \{\mathsf{bad}\} \rangle$. We have that $\phi \not\sqsubseteq_s \psi$ if, and only if, Π has a solution of length at most $(N+1) \times \mathsf{depth}(\phi) + 1$ where N is the number of names n occurring in ϕ at a key position, i.e. such that n (resp. $\mathsf{pub}(n)$) occurs in key position of an encryption in ϕ.

Intuitively, once all needed keys are derived, the minimal plan witnessing non-inclusion contains at most $\mathsf{depth}(\phi)$ rules where $\mathsf{depth}(\phi)$ is the maximal depth of a term occurring ϕ. Then we may need $\mathsf{depth}(\phi)$ rule to derive each deducible key, hence the bound.

4 From trace inclusion to planning

We are now ready for the active case. Given two configurations, we show how to build a planning problem such that the planning problem has a solution if, and only if, the two corresponding configurations are not in trace inclusion.

In several places of this section, we will consider three special constants, namely $c_0^\star$ and $c_1^\star$ of sort atom, and $c_+^\star$ of sort $\mathsf{bitstring}$. These three constants have a special type, denoted $\tau_\star$.

4.1 Abstract protocol rules

We first define the abstract rules describing the protocol behaviour. We denote $\mathcal{C}_{\mathcal{P}}$ (resp. $\mathcal{C}_{\mathcal{Q}}$) the constants from $\mathcal{C}_0$ occurring in $\mathcal{P}$ (resp. $\mathcal{Q}$), and we consider $\mathcal{C}^\star = (\mathcal{C}_{\mathcal{P}} \cup \mathcal{C}_{\mathcal{Q}}) \uplus \{c_0^\star, c_1^\star, c_+^\star\}$. For simplicity we assume that variables of $\mathcal{P}$ and $\mathcal{Q}$ are disjoint. In addition to the facts of the form $\mathsf{att}(u, v)$ used to represent attacker's knowledge, we also consider:

- facts of the form $\mathsf{Phase}(i)$ with $i \in \mathbb{N}$ to represent phases; and
- facts of the form $\mathsf{St}(P, Q) = \mathsf{state}^c_{P,Q}(id_P, id_Q)$ where P, Q are two basic processes on channel c, and id_P (resp. id_Q) is the identity substitution of domain $fv(P)$ (resp. $fv(Q)$).

Therefore, in this section, we consider the infinite set of facts $\mathcal{F}act_0$ that consists of all the ground facts of this form, plus the special symbol bad.

To deal with phases, we mimic the PHASE rule by considering basic processes in normal form w.r.t. the rule $i{:}j{:}P \to j{:}P$. Then, the transformation $\mathsf{Rule}(P; Q)$ from basic processes (in normal form) to abstract planning rules is defined by $\mathsf{Rule}(P; Q) = \emptyset$ when $P = i{:}0$, and otherwise:

1. Case output: i.e. if $P = i{:}\mathsf{out}(c, u).P'$.
   - $\{\mathsf{St}(P, Q), \mathsf{Phase}(i) \to \mathsf{att}(u, v), \mathsf{St}(P', Q');\ \mathsf{St}(P, Q)\} \cup \mathsf{Rule}(i{:}P'; i{:}Q')$ when if $Q = i{:}\mathsf{out}(c, v).Q'$
   - $\{\mathsf{St}(P, Q), \mathsf{Phase}(i) \to \mathsf{att}(u, c_0^\star),\ \mathsf{bad}\}$ otherwise.
2. Case input: i.e. $P = i{:}\mathsf{in}(c, u).P'$.
   - $\{\mathsf{St}(P, Q), \mathsf{att}(u, v), \mathsf{Phase}(i) \to \mathsf{St}(P', Q');\ \mathsf{St}(P, Q)\} \cup \mathsf{Rule}(i{:}P'; i{:}Q')$ when $Q = i{:}\mathsf{in}(c, v).Q'$
   - $\{\mathsf{St}(P, Q), \mathsf{att}(u, x), \mathsf{Phase}(i) \to \mathsf{bad}\}$ otherwise (with x fresh).

Intuitively, abstract rules simply try to mimic each step of P by a similar step in Q. Clearly, if Q cannot follow P, the two processes are not in trace equivalence, which is modelled here by the bad state. Note that, in case $P = i{:}\mathsf{out}(c, u).P'$ whereas Q is not ready to perform an output, bad will be triggered only if the sent term is indeed a message. This transformation is then extended to protocols in a natural way considering in addition planning rule to model phase changes. We consider $\mathcal{P} = \{P_1, \ldots, P_n\}$ and $\mathcal{Q} = \{Q_1, \ldots, Q_n\}$, and we assume w.l.o.g. that P_i and Q_i are basic processes on channel c_i. We define:

- $\mathsf{Rule}(\mathcal{P}, \mathcal{Q}) = \mathsf{Rule}(P_1, Q_1) \cup \ldots \cup \mathsf{Rule}(P_n, Q_n)$.
- $\mathcal{R}^{\mathsf{phase}} = \{\mathsf{Phase}(i) \to \mathsf{Phase}(i + 1)\,;\, \mathsf{Phase}(i) \mid i \in \mathbb{N}\}$.

4.2 Concrete protocol rules

To derive concrete rules from the abstract ones, we could instantiate them with arbitrary terms. However, this would not allow us to derive a decision procedure. Moreover, we would like our algorithm to have good performance. To achieve this, we first show that only three constants need to be considered (and no nonces), in addition to those explicitly mentioned in the protocol.

Given a protocol $\mathcal{P}$ that is type-compliant w.r.t. to a typing system $(\mathcal{T}_\mathcal{P}, \delta_\mathcal{P})$ (and such that $\tau_\star$ does not occur in $\delta_\mathcal{P}(\mathcal{P})$), an execution $\mathcal{P} \xrightarrow{\mathsf{tr}} (\mathcal{P}'; \phi'; \sigma'; i')$ is *quasi-well-typed* if $\delta_\mathcal{P}(x\sigma') \preceq \delta_\mathcal{P}(x)$ for every variable $x \in \mathit{dom}(\sigma')$ where $\preceq$ is the smallest relation on types defined as follows:

- $\tau_\star \preceq \tau$ and $\tau \preceq \tau$ for any type τ (initial or not);
- $\mathsf{f}(\tau_1, \ldots, \tau_k) \preceq \mathsf{f}(\tau'_1, \ldots, \tau'_k)$ when $\tau_1 \preceq \tau'_1, \ldots, \tau_k \preceq \tau'_k$, and $\mathsf{f} \in \Sigma_\mathsf{c}$.

The attacker needs at most the constants $c_0^\star, c_1^\star, c_+^\star$ to mount an attack.

Theorem 1. *Let $\mathcal{K}_\mathcal{P}$ be an initial configuration type-compliant w.r.t. $(\mathcal{T}_\mathcal{P}, \delta_\mathcal{P})$ and $\mathcal{K}_\mathcal{Q}$ be another initial $\mathcal{C}_0$-configuration. We have that $\mathcal{K}_\mathcal{P} \not\sqsubseteq_t \mathcal{K}_\mathcal{Q}$ if, and only if, there exists a witness $(\mathsf{tr}, \phi) \in \mathsf{trace}(\mathcal{K}_\mathcal{P})$ of this non-inclusion which only involves constants from $\mathcal{C}^\star$, simple recipes, and with a quasi-well-typed underlying execution.*

The existence of a quasi well-typed witness comes from [14] with some extra work to guarantee that we can consider simple recipes. The reduction to three constants extends the previous reduction [16] to asymmetric primitives.

Flattening. In terms of efficiency, one key step of our algorithm is to avoid composition rules from the attacker. For static inclusion, we only consider specific contexts, hence very specific synthesis rules, guided by the form of the underlying frames. For the active case, we transform protocol rules in order to pre-compute all necessary composition steps. This flattening step was already used in *e.g.* [3, 16], and is quite intuitive.

Example 7. Consider our Denning Sacco protocol presented in Example 1. Agent B expects a message of the form $u = \{\mathsf{sign}(\langle \mathsf{a}, \mathsf{b}, x\rangle_3, sk_a)\}_{\mathsf{pub}(sk_b)}$. Either the attacker obtains a message m of the expected form, or the attacker obtains several components of it and forges the whole message. For example, it is sufficient for him to obtain m_1 of the form $u_1 = \mathsf{sign}(\langle \mathsf{a}, \mathsf{b}, x\rangle_3, sk_a)$ and m_2 of the form $u_2 = \mathsf{pk}(sk_b)$. Therefore, in addition to the (informal) protocol rule $u \to \ldots$, we also consider the rule $u_1, u_2 \to \ldots$. Similarly, we also need to consider the rules $\mathsf{a}, \mathsf{b}, x, sk_a, \mathsf{pk}(sk_b) \to \ldots$ and $\mathsf{a}, \mathsf{b}, x, sk_a, sk_b \to \ldots$.

More generally, given an abstract protocol rule r, we now define $\mathsf{Flat}(r)$ the set of rules obtained by performing flattening on each fact. To decompose a term, we follow its structure, and the structure of a variable is given by its type. Moreover, when the other side of the process is not able to follow the decomposition, this leads us to a failure rule.

Definition 8. *Given a term $u \in \mathcal{T}_0(\Sigma_\mathsf{c}, \mathcal{C}_0 \uplus \mathcal{N} \uplus \mathcal{X})$, we say that u is* decomposable *when either $u \in \mathcal{X}$ and $\delta_\mathcal{P}(u)$ is not an initial type; or $u \notin \mathcal{C}_0 \uplus \mathcal{N} \uplus \mathcal{X}$.*

A variable of non initial 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, and let $\mathsf{f} \in \Sigma_\mathsf{c}$ be such that $\delta_\mathcal{P}(u) = \mathsf{f}(\tau_1, \ldots, \tau_k)$, $\mathsf{split}(\mathsf{att}(u, v)) = (\mathsf{f}; \{\mathsf{att}(x_1, y_1), \ldots, \mathsf{att}(x_k, y_k)\}; \sigma_\mathcal{P}; \sigma_\mathcal{Q})$ where

- $x_1, \ldots, x_k$ are fresh variables of type $\tau_1, \ldots, \tau_k$, $\sigma_{\mathcal{P}} = mgu(u, \mathsf{f}(x_1, \ldots, x_k))$;
- $y_1, \ldots, y_k$ are fresh variables, $\sigma_{\mathcal{Q}} = mgu(v, \mathsf{f}(y_1, \ldots, y_k))$.

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

Let r be an abstract rule of the form $\mathsf{Pre} \to \mathsf{Add}; \mathsf{Del}$ with $f = \mathsf{att}(u, v) \in \mathsf{Pre}$ 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 $\mathsf{decom}(r, f)$, is defined as follows:

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

Then, decomposition is applied recursively on each rule.

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

Concretization. Given an abstract rule r, we denote $vars_{\mathsf{left}}(r)$ the variables occurring on the left (first parameter) of a predicate occurring in r, i.e.

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

Given a substitution σ grounding for r, the application of σ on an abstract state is the concrete state obtained by simply composing the substitutions, i.e.
$$st^c_{P,Q}(\sigma_P, \sigma_Q)\sigma = st^c_{P,Q}(\sigma \circ \sigma_P, \sigma \circ \sigma_Q).$$

Given an abstract protocol rule r, its concretizations $\mathsf{Concrete}(r)$ simply consist in all its positive and negative concretizations. The positive concretizations are all its instantiations that are quasi-well-typed w.r.t. the left side of the rule.

$$\mathsf{Concrete}^+(r) = \{r\sigma \mid \sigma \text{ substitution such that } r\sigma \text{ only involves messages with constants in } \mathcal{C}^\star \text{ and } \delta_{\mathcal{P}}(x\sigma) \preceq \delta_{\mathcal{P}}(x) \text{ for any } x \in vars_{\mathsf{left}}(r)\}$$

Similarly to the static case, we need to make sure that we can detect when P and Q are *not* in trace inclusion, and we therefore consider some additional rules. Given an abstract protocol rule $r = \mathsf{Pre} \to \mathsf{Add}; \mathsf{Del}$, $\mathsf{Concrete}^-(r)$ is the set of planning rules that contains: $f_1, \ldots, f_k \to \mathsf{bad}$ for any sequence of facts $f_1, \ldots, f_k$ such that $f_1, \ldots, f_k$ left-unify with Pre with substitution σ_L and $u \in \mathcal{T}_0(\Sigma_{\mathsf{c}}, \mathcal{N} \cup \mathcal{C}^\star)$ for any $\mathsf{att}(u, v) \in \mathsf{Add}\sigma_L$, 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 $v \notin \mathcal{T}_0(\Sigma_{\mathsf{c}}, \mathcal{N} \cup \mathcal{C}^\star)$ for some $\mathsf{att}(u, v) \in \mathsf{Add}\sigma_R$.

Main result Our main technical result states that our encoding is sound and complete: two protocols are in trace inclusion if, and only if, the corresponding planning system has a solution. Moreover, when a witness of non-inclusion exists, we are able to bound the length of the resulting plan. Below, $\mathsf{nb}_{\mathsf{in}}(\mathcal{P})$ (resp. $\mathsf{nb}_{\mathsf{out}}(\mathcal{P})$) denotes the number of inputs (resp. outputs) occurring in $\mathcal{P}$ whereas $\mathsf{max}_{\mathsf{phase}}(\mathcal{P})$ is the maximal integer occurring in a phase instruction in $\mathcal{P}$.

Theorem 2. *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:*

$$\mathsf{Concrete}(\mathsf{R}_{\mathsf{Ana}} \cup \mathsf{Flat}(\mathsf{Rule}(\mathcal{P}, \mathcal{Q}))) \cup \mathcal{R}^{\mathsf{phase}} \cup \mathcal{R}^{\mathsf{test}_1}_{\mathsf{fail}} \cup \mathcal{R}^{\mathsf{test}_2}_{\mathsf{fail}} \cup \mathcal{R}^{\mathsf{atom}}_{\mathsf{fail}} \cup \mathcal{R}^{\mathsf{check}}_{\mathsf{fail}} \cup \mathcal{R}^{\mathsf{pub}}_{\mathsf{fail}}$$

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

$$1 + \mathsf{nb}_{\mathsf{in}}(\mathcal{P}) + \mathsf{nb}_{\mathsf{out}}(\mathcal{P}) + \mathsf{max}_{\mathsf{phase}}(\mathcal{P}) + \mathsf{depth}(\delta_\mathcal{P}(\mathcal{P})) \times [1 + \mathsf{nb}_{\mathsf{in}}(\mathcal{P}) + N]$$

where N is the number of names occurring in $\mathcal{P}$ having a key type, i.e. such that $\delta_\mathcal{P}(n)$ (resp. $\mathsf{pub}(\delta_\mathcal{P}(n))$) occurs in key position of an encryption in $\delta_\mathcal{P}(\mathcal{P})$.

Proof. (Sketch) It is rather easy to establish that a solution to the planning problem defines a witness of non trace inclusion. Conversely, thanks to Theorem 1, if $\mathcal{P} \not\sqsubseteq_t \mathcal{Q}$, then there exists a quasi well-typed witness of non trace inclusion, that uses at most three constants (besides the constants of $\mathcal{P}$ and $\mathcal{Q}$). This witness guides the definition of a plan of Π. Establishing a not too coarse bound on its length requires some care. It relies on the flattening of the protocol and the fact that the plan can mimic the computation of several messages in parallel. □

5 Algorithm

Similarly to the algorithm presented in [16], we decide trace inclusion by applying graph planning and SAT-solving techniques to the planning problem that encodes trace inclusion (thanks to Theorem 2). Given a protocol $\mathcal{P}$, type-compliant w.r.t. $(\mathcal{T}_\mathcal{P}, \delta_\mathcal{P})$, and a protocol $\mathcal{Q}$, our algorithm proceeds as follows.

1. It first computes the corresponding abstract rules, namely $\mathsf{Flat}(\mathsf{Rule}(\mathcal{P}; \mathcal{Q})) \cup \mathsf{R}_{\mathsf{Ana}}$ and the initial state $\mathsf{Fact}(\mathcal{P}, \mathcal{Q})$.
2. It then applies a planning graph algorithm, a standard technique to solve planning problems (see e.g. [7]). The only difference is that, for efficiency reasons, we do not contruct the planning problem Π *a priori* but instead, we compute it "on the fly", while building the associated planning graph. This planning graph over-approximates the possible solutions by executing several actions in parallel, even if they may be incompatible. Some incompatibilities are recorded and propagated through so-called mutex. The planning graph is deemed to capture all possible plans. More precisely, the planning graph built until depth k captures all possible plans of length at most k.
3. In case no fact bad has been reached while building the planning graph, we can immediately conclude that $\mathcal{P} \sqsubseteq_t \mathcal{Q}$. Otherwise, since the planning graph over-approximates the possible executions, we need to check that bad is truly reachable. This is done by encoding each path leading to bad as a SAT formula. We then call the SAT solver mini-SAT to decide its satisfiability. In case bad is indeed reachable, mini-SAT provides a solution that is translated back to a witness of non-inclusion. To improve termination, we check accessibility of a state containing bad as soon as it appears in the graph, even if the construction of the graph is not completed yet.

Termination. The algorithm defined above may not terminate. The planning graph contains facts of the form $\mathsf{att}(u, v)$ where u must be (quasi) well-typed. There is therefore only a finite number of such u. However, the planning graph construction may introduce several facts of the form $\mathsf{att}(u, v_1), \ldots, \mathsf{att}(u, v_k)$, where the v_i get arbitrarily large. We exhibit some (contrived) examples where the algorithm does not terminate (see Appendix A). [16] suggests that termination could be enforced by checking at each step (thanks to the SAT-solver) that each node of the planning graph is indeed reachable. This would however not be practical. Instead, we can enforce termination thanks to the bound provided in Theorem 2 that also bounds the maximal depth of the planning graph that needs to be considered. Indeed, it is sufficient to simply stop the construction of the planning graph as soon as the bound is reached. The interest of this approach is that we guarantee termination at no cost (computing the bound is immediate). In practice, the planning graph is typically much smaller than this bound.

SAT-Equiv. We have implemented our new algorithm in the tool SAT-Equiv, extending it to protocols with phases and all the standard cryptographic primitives and guaranteeing termination. Moreover, we significantly improve its efficiency by rewriting parts of the codes and modifying the data structure.

6 Experiments

In this section, we analyse several protocols of the literature and compare the results obtained using different tools. We ran our experiments a single Intel 3.1 GHz Xeon. We limit the memory to 128 Go (MO stands for memory out) and the execution time to 24h (TO stands for time out).

For all the considered protocols, we analyse strong secrecy of the exchanged key or nonce, as for Example 5, except for the passport protocol (PA), for which we prove anonymity as in [2]. We progressively increase the number of sessions in order to consider *a semi complete scenario*, where Alice's role is instantiated by honest a talking to honest b or dishonest c and Bob's role is instantiated by b talking to a or c. This typically corresponds to 7 sessions in the case of a symmetric key protocol (with 3 roles).

6.1 Comparison with the other tools

Our experiments show a significant speed-up w.r.t. the original version of SAT-Equiv [16]. Our new is 100 faster in average, allowing to analyse about twice more sessions, as exemplified in Figure 2 . We compare SAT-Equiv with other tools of the literature that decide equivalence for a bounded number of sessions, namely Spec [25], Akiss [8] and Deepsec [11]. We did not include APTE in our study [9] as it is now subsumed by Deepsec. For each protocol, we progressively increased the number of sessions until we reached a time out. The overall results of our experiments are summarized in Figure 2. They show a significant speed-up even w.r.t. the very recent Deepsec tool. Note however that Deepsec covers more

	Spec	Akiss	Deepsec	CSF'17	Sat-Eq
Denning-Sacco	7	10	35	98	> 210 (4h)
Needham-Schroeder sym	6	6	21	21	94^* (20h30)
Wide Mouth Frog	7	12	28	84	> 210 (6min)
Yahalom-Paulson	6	6	12	7	> 28 (7h)
Passive Authentication	6	8	46	–	> 400 (98s)
Active Authentication	6	8	50	–	> 400 (78s)
Needham-Schroeder-Lowe	4	6	16	–	> 64 (11min)
Denning-Sacco signature	8	8	18	–	> 64 (100s)

Fig. 2. Comparison of SAT-Equiv with the other tools. We indicate the number of sessions for which the tool fails (time out, memory out, or other issues). When we did not reach the limit of the tool, we write $> k$ to indicate that the tool can analyse more than k sessions, and we indicate the analysis time for k. * see Section 6.2

protocols (with else branches, or not type compliant), except if they include phases. Deepsec can also be parallelized thus the analysis time can be divided by the number of available cores. The detailed results for the Denning-Sacco protocol are below.

Denning-Sacco	Spec	Akiss	Deepsec	CSF'17	SAT-Equiv	
3	12 s	0.08 s	<0.01 s	0.3 s	0.07 s	42
6	5 h	9 s	<0.01 s	1 s	0.1 s	64
7	MO	75 s	<0.01 s	2 s	0.2 s	74
10		MO	0.01 s	4 s	0.3 s	114
21			18 s	60 s	1.3 s	216
35			TO	9 min	6 s	344
84				13 h	164 s	792
98				TO	6 min	920
210					4h20	1942

The 2nd column for SAT-Equiv indicates the theoretical bound on the length of the planning graph, as given by Theorem 2. This illustrates that this bound remains reasonable although our tool actually terminates before reaching it.

6.2 Towards an unbounded number of sessions

Although equivalence is undecidable in general for an unbounded number of sessions, [13] exhibits a decidability result, for type-compliant protocols that have an *acyclic* dependency graph. Intuitively, the dependency graph captures how a message expected as input may be built (and therefore may depend) from messages sent as output of the protocol. Decidability is proven by showing that a (minimal) attack trace may be mapped to this dependency graph. Looking at the dependency graphs of the Denning-Sacco and the Needham-Schroeder symmetric key protocols, we deduce that it is sufficient to analyse respectively 42 and 94 sessions. Thanks to the efficiency of SAT-Equiv, we can easily analyse 42 sessions of Denning-Sacco (in 10s). We can therefore deduce from [13] that the protocol remains secure even if the considered sessions are arbitrarily replicated. The case of the Needham-Schroeder protocol requires a bit more work as 94 sessions is slightly out of reach of SAT-Equiv. However, we noticed that, according to [13], we do not need to analyse 94 full sessions. Instead, some of them may be truncated (a minimal attack will use only the first step for example). Since SAT-Equiv can prove equivalence of these refined 94 sessions (in 20h30min), we can again deduce from [13] that the protocol remains secure even if the considered sessions are arbitrarily replicated.

As future work, we plan to optimize the bound on sessions induced by [13] and automatically generate the desired scenario, in order to extend SAT-Equiv to proofs of equivalence for an unbounded number of sessions.

References

1. Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *Proc. 28th ACM Symposium on Principles of Programming Languages*, POPL '01, pages 104–115. ACM, 2001.
2. Myrto Arapinis, Tom Chothia, Eike Ritter, and Mark Ryan. Analysing unlinkability and anonymity using the applied pi calculus. In *Proc. 23rd Computer Security Foundations Symposium (CSF'10)*, pages 107–121. IEEE Computer Society Press, 2010.
3. Alessandro Armando and Luca Compagna. Sat-based model-checking for security protocols analysis. *International Journal of Information Security*, 7:3–32, 2008.
4. Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. Verified models and reference implementations for the TLS 1.3 standard candidate. In *Proc. 38th IEEE Symposium on Security and Privacy (S&P'17)*. IEEE Computer Society Press, 2017.
5. Bruno Blanchet. Symbolic and computational mechanized verification of the ARINC823 avionic protocols. In *Proc. 30th IEEE Computer Security Foundations Symposium (CSF'17)*, pages 68–82. IEEE Computer Society Press, 2017.
6. Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated Verification of Selected Equivalences for Security Protocols. *Journal of Logic and Algebraic Programming*, 75(1):3–51, February–March 2008.
7. Avrim Blum and Merrick Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300, 1997.
8. Rohit Chadha, Ştefan Ciobâcă, and Steve Kremer. Automated verification of equivalence properties of cryptographic protocols. In *Proc. 21th European Symposium on Programming (ESOP'12)*, LNCS, pages 108–127, 2012.
9. Vincent Cheval. Apte: an algorithm for proving trace equivalence. In *Proc. 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14)*, volume 8413 of *LNCS*, pages 587–592, 2014.
10. Vincent Cheval, Véronique Cortier, and Stéphanie Delaune. Deciding equivalence-based properties using constraint solving. *Theoretical Computer Science*, 492:1–39, June 2013.
11. Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. Deepsec: Deciding equivalence properties in security protocols - theory and practice. In *Proc. 39th IEEE Symposium on Security and Privacy (S&P'18)*, pages 525–542. IEEE Computer Society Press, 2018.

12. Rémy Chrétien, Véronique Cortier, and Stéphanie Delaune. Typing messages for free in security protocols: the case of equivalence properties. In *Proc. 25th International Conference on Concurrency Theory (CONCUR'14)*, LNCS, Rome, Italy, 2014. Springer.
13. Rémy Chrétien, Véronique Cortier, and Stéphanie Delaune. Decidability of trace equivalence for protocols with nonces. In *Proc. the 28th IEEE Computer Security Foundations Symposium (CSF'15).* IEEE Computer Society Press, 2015.
14. Rémy Chrétien, Véronique Cortier, Antoine Dallon, and Stéphanie Delaune. Typing messages for free in security protocols. Technical report, 2018.
15. Luca Compagna. *SAT-based Model-Checking of Security Protocols.* PhD thesis, Università degli Studi di Genova and the University of Edinburgh (joint programme), September 2005.
16. Véronique Cortier, Antoine Dallon, and Stéphanie Delaune. SAT-Equiv: an efficient tool for equivalence properties. In *Proc. of the 30th IEEE Computer Security Foundations Symposium (CSF'17).* IEEE Computer Society Press, August 2017.
17. Véronique Cortier, Antoine Dallon, and Stéphanie Delaune. Efficiently deciding equivalence for standard primitives and phases. Research report, `https://hal.archives-ouvertes.fr/hal-01819366`, June 2018.
18. Véronique Cortier, Niklas Grimm, Joseph Lallemand, and Matteo Maffei. A type system for privacy properties. In *Proc. 24th ACM Conference on Computer and Communications Security (CCS'17)*, pages 409–423. ACM, 2017.
19. Véronique Cortier, Niklas Grimm, Joseph Lallemand, and Matteo Maffei. Equivalence properties by typing in cryptographic branching protocols. In *Proc. 7th International Conference on Principles of Security and Trust (POST'18)*, LNCS, 2018.
20. Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. A comprehensive symbolic analysis of TLS 1.3. In *Proc. 24th ACM Conference on Computer and Communications Security (CCS'17)*, pages 1773–1788. ACM, 2017.
21. Stéphanie Delaune, Steve Kremer, and Mark Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, (4):435–487, July 2008.
22. Santiago Escobar, Catherine Meadows, and José Meseguer. A rewriting-based inference system for the NRL protocol analyzer and its meta-logical properties. *Theoretical Computer Science*, 367(1-2):162–202, 2006.
23. Henri Kautz and Bart Selman. Planning as satisfiability. In *Proc. 10th European Conference on Artificial Intelligence (ECAI'92)*, pages 359–363, 1992.
24. Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In *Proc. 25th International Conference on Computer Aided Verification (CAV'13)*, volume 8044 of *LNCS*, pages 696–701. Springer, 2013.
25. Alwen Tiu and Jeremy Dawson. Automating open bisimulation checking for the spi calculus. In *Proc. 23rd IEEE Computer Security Foundations Symposium (CSF'10)*, pages 307–321. IEEE Computer Society Press, 2010.

A Examples of non termination

We exhibit two examples on which the original SAT-Equiv algorithm does not terminate. Given a channel c, consider $P(c)$ and $Q(c)$ defined as follows.

$$P(c) := \mathsf{in}(c, \langle x, \mathsf{a}\rangle).\mathsf{out}(c, \langle x, \mathsf{a}\rangle)$$
$$Q(c) := \mathsf{in}(c, \langle x, \mathsf{a}\rangle).\mathsf{out}(c, \langle\langle x, x\rangle, \mathsf{a}\rangle)$$

where a is a public constant and x a variable. We consider $\mathcal{K}_P = \{P(c_1); P(c_2)\}$ and $\mathcal{K}_Q = \{Q(c_1); Q(c_2)\}$ for some public channel names c_1, c_2. Starting with $\mathsf{att}(\mathsf{b}, \mathsf{b})$ (with b a public constant in the initial knowledge of the attacker), the following facts will be successively added when computing the planning graph:

$$\mathsf{att}(\langle \mathsf{b}, \mathsf{a}\rangle, \langle \mathsf{b}, \mathsf{a}\rangle),\ \mathsf{att}(\langle \mathsf{b}, \mathsf{a}\rangle, \langle\langle \mathsf{b}, \mathsf{b}\rangle, \mathsf{a}\rangle),\ \mathsf{att}(\langle \mathsf{b}, \mathsf{a}\rangle, \langle\langle\langle \mathsf{b}, \mathsf{b}\rangle, \langle \mathsf{b}, \mathsf{b}\rangle\rangle, \mathsf{a}\rangle), \ldots$$

Actually, $\mathsf{att}(\langle \mathsf{b}, \mathsf{a}\rangle, \langle\langle \mathsf{b}, \mathsf{b}\rangle, \mathsf{a}\rangle)$ can be added in two different ways: either considering the output on c_1, or the one on c_2. Therefore this fact will not be put in mutex with the other ones. In particular, the fact $\mathsf{att}(\langle \mathsf{b}, \mathsf{a}\rangle, \langle\langle \mathsf{b}, \mathsf{b}\rangle, \mathsf{a}\rangle)$ and the state fact indicating that the process on channel c_1 has not yet started are not in mutex, and can be used to trigger the planning rules leading to $\mathsf{att}(\langle \mathsf{b}, \mathsf{a}\rangle, \langle\langle\langle \mathsf{b}, \mathsf{b}\rangle, \langle \mathsf{b}, \mathsf{b}\rangle\rangle, \mathsf{a}\rangle)$. Since the term computed on the Q's side grows at each step, this computation is endless.

Here, $\mathcal{K}_P$ is not trace included in $\mathcal{K}_Q$: an attacker can distinguish between b and $\langle \mathsf{b}, \mathsf{b}\rangle$. So, as soon as a message is outputted, the resulting frames are not in static inclusion. Therefore, termination can be retrieved by enforcing SAT-Equiv to stop the exploration of the planning graph as soon as an attack is found.

We can turn this example into a more complex one on which the original SAT-Equiv will not terminate even if we decide to stop the exploration of the planning graph as soon as an attack is found. Consider the processes $P_0(c)$, $P_1(c)$ and $Q_1(c)$ given below. We assume that k is name representing a symmetric secret key, whereas $\mathsf{a}, \mathsf{b}, \mathsf{c}$ are public constants.

$$P_0(c) = \mathsf{in}(c, x).\mathsf{out}(c, \mathsf{senc}(x, k))$$
$$P_1(c) = \mathsf{in}(c, \langle \mathsf{senc}(\mathsf{a}, k), \mathsf{senc}(\mathsf{b}, k), \mathsf{senc}(\mathsf{c}, k)\rangle_3).P(c)$$
$$Q_1(c) = \mathsf{in}(c, \langle \mathsf{senc}(\mathsf{a}, k), \mathsf{senc}(\mathsf{b}, k), \mathsf{senc}(\mathsf{c}, k)\rangle_3).Q(c)$$

We consider the configurations $\mathcal{K}'_P = \{P_0(c_0); P_0(c_1); P_1(c_2); P_1(c_3)\}$ and $\mathcal{K}'_Q = \{P_0(c_0); P_0(c_1); Q_1(c_2); Q_1(c_3)\}$ where c_0, c_1, c_2, c_3 are public channel names. Processes P_0 on channels c_0 and c_1 are used as oracles. Roughly, we can get two ciphertexts among the three ciphertexts: $\mathsf{senc}(\mathsf{a}, k)$, $\mathsf{senc}(\mathsf{b}, k)$, and $\mathsf{senc}(\mathsf{c}, k)$. It is however not possible to get the three of them. Noticing this, it is then easy to see that $\mathcal{K}_P$ and $\mathcal{K}_Q$ are trace included.

However, as in the previous example, the planning graph is not precise enough to detect that it is not possible to obtain these three ciphertexts. Once the inputs on channel c_2 and c_3 are executed, we reach a situation similar to the one discussed in the previous example. Each time bad will be added into the planning graph, our SAT encoding will tell us that this state is not truly reachable (but only exists in the over-approximation). Thus, we will continue to explore the planning graph for ever since no attack will be found (the protocols are trace-equivalent).