Typing messages for free in security protocols: the case of equivalence properties ⋆

Rémy Chrétien[1,2], Véronique Cortier[1], and Stéphanie Delaune[2]

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

Abstract. Privacy properties such as untraceability, vote secrecy, or anonymity are typically expressed as behavioural equivalence in a process algebra that models security protocols. In this paper, we study how to decide one particular relation, namely trace equivalence, for an unbounded number of sessions.
Our first main contribution is to reduce the search space for attacks. Specifically, we show that if there is an attack then there is one that is well-typed. Our result holds for a large class of typing systems and a large class of determinate security protocols. Assuming finitely many nonces and keys, we can derive from this result that trace equivalence is decidable for an unbounded number of sessions for a class of tagged protocols, yielding one of the first decidability results for the unbounded case. As an intermediate result, we also provide a novel decision procedure in the case of a bounded number of sessions.

1 Introduction

Privacy properties such as untraceability, vote secrecy, or anonymity are typically expressed as behavioural equivalence (*e.g.* [9, 5]). For example, the anonymity of Bob is typically expressed by the fact that an adversary should not distinguish between the situation where Bob is present and the situation where Alice is present. Formally, the behaviour of a protocol can be modelled through a process algebra such as CSP or the pi calculus, enriched with terms to represent cryptographic messages. Then indistinguishability can be modelled through various behavioural equivalences. We focus here on trace equivalence, denoted $\approx$. Checking for privacy then amounts into checking for trace equivalence between processes, which is of course undecidable in general. Even in the case of a bounded number of sessions, there are few decidability results and the associated decision procedures are complex [6, 22, 11]. In this paper, we study trace equivalence in the case of an unbounded number of sessions.

Our contribution. Our first main contribution is a simplification result, that reduces the search space for attacks: if there is an attack, then there exists a well-typed attack. More formally, we show that if there is a witness (*i.e.* a trace) that $P \not\approx Q$ then there exists a witness which is well-typed w.r.t. P or Q, provided that P and Q are determinate

⋆ The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement n° 258865, project ProSecure, and the ANR project JCJC VIP n^o 11 JS02 006 01.

processes (intuitively, messages that are outputted are completely determined by the interactions of the protocol with the environment, *i.e.* the attacker). This typing result holds for an unbounded number of sessions and an unbounded number of nonces, that is, it holds even if P and Q contain arbitrary replications and NEW operations. It holds for any typing system provided that any two unifiable encrypted subterms of P (or Q) are of the same type. It is then up to the user to adjust the typing system such that this hypothesis holds for the protocols under consideration. For simplicity, we prove this typing result for the case of symmetric encryption and concatenation but we believe that our result could be extended to the other standard cryptographic primitives.

The finer the typing system is, the more our typing result restricts the attack search. In general, our typing result does not yield directly a decidability result since even the simple property of reachability is undecidable for an unbounded number of sessions and arbitrary nonces, even if the messages are of bounded size (*e.g.* [3]). Indeed, our typing system ensures the existence of a well-typed attack (if any) but the number of well-typed traces may remain infinite. To obtain decidability, we further assume a finite number of terms of each type (*i.e.* in particular a finite number of nonces). Decidability of trace equivalence then follows from our main typing result, for a class of *simple* protocols where each subprocess uses a distinct channel (intuitively, a session identifiers).

As an application, we consider the class of tagged protocols introduced by Blanchet and Podelski [8]. An easy way to achieve this in practice by labelling encryption and is actually a good protocol design principle [2, 19]. We show that tagged protocols induce a typing system for which trace equivalence is decidable, for simple protocols and for an unbounded number of sessions (but a fixed number of nonces).

Interestingly, the proof of our main typing result involves providing a new decision procedure for trace equivalence in the case of a bounded number of sessions. This is a key intermediate result of our proof. Trace equivalence was already shown to be decidable for a bounded number of sessions (*e.g.* [22, 11]) but we propose a novel decision procedure that further provides a *well-typed* witness whenever the two processes are not in trace equivalence. Compared to existing procedures (*e.g.* [22]), we show that it is only necessary to consider unification between encrypted terms. We believe that this new procedure is of independent interest since it reduces the number of traces (executions) that need to be considered. Our result could therefore be used to speed up equivalence checkers like SPEC [22]. Detailed proofs of our results can be found in [14].

Related work. Formal methods have been very successful for the analysis of security protocols and many decision procedures and tools (*e.g.* [21, 20, 16]) have been proposed. However, most of these results focus on reachability properties such as confidentiality or authentication. Much fewer results exist for behavioural equivalences. Based on a procedure proposed by Baudet [6], a first decidability result has been proposed for determinate process without else branches, and for equational theories that capture most standard primitives [12]. Then Tiu and Dawson [22] have designed and implemented a procedure for open bisimulation, a notion of equivalence stronger than the standard notion of trace equivalence. Cheval *et al* [11] have proposed and implemented a procedure for processes with else branches and standard primitives. The tool AkisS [10] is also dedicated to trace equivalence but is not guaranteed to terminate. However, all these results focus on a bounded number of sessions. An exception is the

tool ProVerif which can handle observational equivalence for an unbounded number of sessions [7]. It actually reasons on a stronger notion of equivalence (which may turn to be too strong in practice) and is again not guaranteed to terminate.

To our knowledge, the only decidability result for an unbounded number of sessions is [13]. It is shown that trace equivalence can be reduced to the equality of languages of pushdown automata. A key hypothesis for reducing to pushdown automata is that protocol rules have at most one variable, that is, at any execution step, any participant knows already every component of the message he received except for at most one component (*e.g.* a nonce received from another participant). Moreover variables shall not occur in key position, *i.e.* agents may not use received keys for encryption. This strongly limits the class of protocols that can be considered and the approach is strictly bound to this "one-variable" hypothesis. In contrast, we can consider here a much wider class of protocols, provided that they are tagged (which is easy to implement).

Our proof technique is inspired from the approach developed by Arapinis *et al* [4] for bounding the size of messages of an attack for the reachability case. Specifically, they show for some class of tagged protocols, that whenever there is an attack, there is a well-typed attack (for a particular typing system). We somehow extend their approach to trace equivalence and more general typing systems.

2 Model for security protocols

Security protocols are modelled through a process algebra inspired from [1] that manipulates terms.

2.1 Syntax

Term algebra. We assume an infinite set $\mathcal{N}$ of *names*, which are used to represent keys and nonces, and two infinite disjoint sets of *variables* $\mathcal{X}$ and $\mathcal{W}$. The variables in $\mathcal{W}$ intuitively refer to variables used to store messages learnt by the attacker. We assume a signature $\mathcal{F}$, *i.e.* a set of function symbols together with their arity. We consider:

$$\Sigma_{\mathsf{c}} = \{\mathsf{enc}, \langle\ \rangle\},\ \Sigma_{\mathsf{d}} = \{\mathsf{dec}, \mathsf{proj}_1, \mathsf{proj}_2\}, \text{ and } \Sigma = \Sigma_{\mathsf{c}} \cup \Sigma_{\mathsf{d}}.$$

The symbols dec and enc of arity 2 represent symmetric decryption/encryption. Pairing is modelled using a symbol of arity 2, denoted $\langle\ \rangle$, and projection functions are denoted proj_1 and proj_2. We further assume an infinite set of *constant symbols* Σ_0 to represent atomic data known to the attacker. The symbols in Σ_{c} are constructors whereas those in Σ_{d} are destructors. Both represent functions available to the attacker.

Given a set of A of atoms (*i.e.* names, variables, and constants), and a signature $\mathcal{F} \in \{\Sigma_{\mathsf{c}}, \Sigma_{\mathsf{d}}, \Sigma\}$, we denote by $\mathcal{T}(\mathcal{F}, \mathsf{A})$ the set of terms built from symbols in $\mathcal{F}$, and atoms in A. The subset of $\mathcal{T}(\Sigma_{\mathsf{c}}, \mathsf{A})$ which only contains terms with atoms as a second argument of the symbol enc, is denoted $\mathcal{T}_0(\Sigma_{\mathsf{c}}, \mathsf{A})$. Terms in $\mathcal{T}_0(\Sigma_{\mathsf{c}}, \Sigma_0 \cup \mathcal{N})$ are called *messages*. An attacker builds his own messages by applying functions to terms he already knows. Formally, a computation done by the attacker is modelled by a term, called a *recipe*, built on the signature Σ using (public) constants in Σ_0 as well as variables in $\mathcal{W}$, *i.e.* a term $R \in \mathcal{T}(\Sigma, \Sigma_0 \cup \mathcal{W})$. Note that such a term does not contain any name.

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*. Two terms u_1 and u_2 are *unifiable* when there exists σ such that $u_1\sigma = u_2\sigma$.

The relations between encryption/decryption and pairing/projections are represented through the three following rewriting rules, yielding a convergent rewrite system:

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

Given $u \in \mathcal{T}(\Sigma, \Sigma_0 \cup \mathcal{N} \cup \mathcal{X})$, we denote by $u{\downarrow}$ its *normal form*. We refer the reader to [18] for the precise definitions of rewriting systems, convergence, and normal forms.

Example 1. Let $s, k \in \mathcal{N}$, and $u = \mathsf{enc}(s, k)$. The term $\mathsf{dec}(u, k)$ models the application of the decryption algorithm on u using k. We have that $\mathsf{dec}(u, k){\downarrow} = s$.

Process algebra. Let $\mathcal{C}h$ be an infinite set of *channels*. We consider processes built using the following grammar where $u \in \mathcal{T}(\Sigma_{\mathsf{c}}, \Sigma_0 \cup \mathcal{N} \cup \mathcal{X})$, $n \in \mathcal{N}$, and $c, c' \in \mathcal{C}h$:

$$P, Q := 0 \mid \mathsf{in}(c, u).P \mid \mathsf{out}(c, u).P \mid (P \mid Q) \mid !P \mid \mathsf{new}\, n.P \mid \mathsf{new}\, c'.\mathsf{out}(c, c').P$$

The process 0 does nothing. The process "$\mathsf{in}(c, u).P$" expects a message m of the form u on channel c and then behaves like $P\sigma$ where σ is a substitution such that $m = u\sigma$. The process "$\mathsf{out}(c, u).P$" emits u on channel c, and then behaves like P. The variables that occur in u are instantiated when the evaluation takes place. The process $P \mid Q$ runs P and Q in parallel. The process $!P$ executes P some arbitrary number of times. The name restriction "$\mathsf{new}\ n$" is used to model the creation in a process of a fresh random number (*e.g.*, a nonce or a key) whereas channel generation "$\mathsf{new}\, c'.\mathsf{out}(c, c').P$" is used to model the creation of a new channel name that shall immediately be made public. Note that we consider only public channels. It is still useful to generate fresh (public) channel names to let the attacker identify the different sessions of a protocol (as it is often the case in practice through sessions identifiers).

We assume that names are implicitly freshly generated, thus $\mathsf{new}\, k.\mathsf{out}(c, k)$ and $\mathsf{out}(c, k)$ have exactly the same behaviour. The construction "new" becomes important in the presence of replication to distinguish whether some value k is generated at each session, *e.g.* in $!(\mathsf{new}\, k.\mathsf{out}(c, k))$ or not, *e.g.* in $\mathsf{new}\ k.(!\mathsf{out}(c, k))$.

For the sake of clarity, we may omit the null process. We also assume that processes are *name and variable distinct*, *i.e.* any name and variable is at most bound once. For example, in the process $\mathsf{in}(c, x).\mathsf{in}(c, x)$ the variable x is bound once and thus the process is name and variable distinct. By contrast, in $\mathsf{in}(c, x) \mid \mathsf{in}(c, x)$, one occurrence of the variable x would need to be renamed. 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.

We assume $\mathcal{C}h = \mathcal{C}h_0 \uplus \mathcal{C}h^{\mathsf{fresh}}$ where $\mathcal{C}h_0$ and $\mathcal{C}h^{\mathsf{fresh}}$ are two infinite and disjoint sets of channels. Intuitively, channels of $\mathcal{C}h^{\mathsf{fresh}}$, denoted $ch_1, \ldots, ch_i, \ldots$ will be used in the semantics to *instantiate* the channels generated during the execution of a protocol. They shall not be part of its specification.

Definition 1. *A* protocol P *is a process such that* P *is ground,* i.e. $fv(P) = \emptyset$*;* P *is name and variable distinct; and* P *does not use channel names from* $\mathcal{C}h^{\mathsf{fresh}}$.

Example 2. The Otway-Rees protocol [15] is a key distribution protocol using symmetric encryption and a trusted server. It can be described informally as follows:

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

where $\{m\}_k$ denotes the symmetric encryption of a message m with key k, A and B are agents trying to authenticate each other, S is a trusted server, K_{as} (resp. K_{bs}) is a long term key shared between A and S (resp. B and S), N_a and N_b are nonces generated by A and B, K_{ab} is a session key generated by S, and M is a session identifier.

We propose a modelling of the Otway-Rees protocol in our formalism. We use restricted channels to model the use of unique session identifiers used along an execution of the protocol. Below, $k_{as}, k_{bs}, m, n_a, n_b, k_{ab}$ are names, whereas a and b are constants from Σ_0. We denote by $\langle x_1, \ldots, x_{n-1}, x_n\rangle$ the term $\langle x_1, \langle \ldots \langle x_{n-1}, x_n\rangle\rangle\rangle$.

$$P_{\mathsf{OR}} = !\,\mathsf{new}\ c_1.\mathsf{out}(c_A, c_1).P_A \mid !\,\mathsf{new}\ c_2.\mathsf{out}(c_B, c_2).P_B \mid !\,\mathsf{new}\ c_3.\mathsf{out}(c_S, c_3).P_S$$

where the processes P_A, P_B are given below, and P_S can be defined in a similar way.

$$P_A = \mathsf{new}\ m.\mathsf{new}\ n_a.\ \mathsf{out}(c_1, \langle m, \mathsf{a}, \mathsf{b}, \mathsf{enc}(\langle n_a, m, \mathsf{a}, \mathsf{b}\rangle, k_{as})\rangle).\ \mathsf{in}(c_1, \langle m, \mathsf{enc}(\langle n_a, x_{ab}\rangle, k_{as})\rangle);$$

$$P_B = \mathsf{in}(c_2, \langle y_m, \mathsf{a}, \mathsf{b}, y_{as}\rangle).\mathsf{new}\ n_b.\mathsf{out}(c_2, \langle y_m, \mathsf{a}, \mathsf{b}, y_{as}, \mathsf{enc}(\langle n_b, y_m, \mathsf{a}, \mathsf{b}\rangle, k_{bs})\rangle).\ \mathsf{in}(c_2, \langle y_m, z_{as}, \mathsf{enc}(\langle n_b, y_{ab}\rangle, k_{bs})\rangle).\mathsf{out}(c_2, \langle y_m, z_{as}\rangle)$$

2.2 Semantics

The operational semantics of a process is defined using a relation over configurations. A *configuration* is a pair $(\mathcal{P}; \phi)$ where:

- $\mathcal{P}$ is a multiset of ground processes.
- $\phi = \{\mathsf{w}_1 \triangleright m_1, \ldots, \mathsf{w}_n \triangleright m_n\}$ 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, *i.e.* terms in $\mathcal{T}_0(\Sigma_{\mathsf{c}}, \Sigma_0 \cup \mathcal{N})$.

We often write P instead of $(\{P\}; \emptyset)$, and $P \cup \mathcal{P}$ or $P \mid \mathcal{P}$ instead of $\{P\} \cup \mathcal{P}$. The terms in ϕ represent the messages that are known by the attacker. The operational semantics of a process is induced by the relation $\xrightarrow{\alpha}$ over configurations defined below.

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

$$(\mathsf{out}(c, u).P \cup \mathcal{P}; \phi) \xrightarrow{\mathsf{out}(c,\mathsf{w}_{i+1})} (P \cup \mathcal{P}; \phi \cup \{\mathsf{w}_{i+1} \triangleright u\}) \quad \text{where } u \text{ is a message and } i \text{ is the number of elements in } \phi$$

$$(\mathsf{new}\ c'.\mathsf{out}(c, c').P \cup \mathcal{P}; \phi) \xrightarrow{\mathsf{out}(c,ch_i)} (P\{^{ch_i}/_{c'}\} \cup \mathcal{P}; \phi) \quad \text{where } ch_i \text{ is the "next" fresh channel name available in } \mathcal{C}h^{\mathsf{fresh}}$$

$$(\mathsf{new}\ n.P \cup \mathcal{P}; \phi) \xrightarrow{\tau} (P\{^{n'}/_{n}\} \cup \mathcal{P}; \phi) \quad \text{where } n' \text{ is a fresh name in } \mathcal{N}$$

$$(!P \cup \mathcal{P}; \phi) \xrightarrow{\tau} (P \cup !P \cup \mathcal{P}; \phi)$$

The first rule allows the attacker to send to some process a term built from publicly available terms and symbols. The second rule corresponds to the output of a term by some process: the corresponding term is added to the frame of the current configuration, which means that the attacker can now access the sent term. 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. The third rule corresponds to the special case of an output of a freshly generated channel name. In such a case, the channel is not added to the frame but it is implicitly assumed known to the attacker, as all the channel names. These three rules are the only observable actions. The two remaining rules are quite standard and are unobservable (τ action) from the point of view of the attacker. The relation $\xrightarrow{\alpha_1 \dots \alpha_n}$ between configurations (where $\alpha_1 \dots \alpha_n$ is a sequence of actions) is defined as the transitive closure of $\xrightarrow{\alpha}$.

Given a sequence of observable actions tr, we write $K \overset{\mathsf{tr}}{\Longrightarrow} K'$ when there exists a sequence $\alpha_1 \dots \alpha_n$ such that $K \xrightarrow{\alpha_1 \dots \alpha_n} K'$ and tr is obtained from $\alpha_1 \dots \alpha_n$ by erasing all occurrences of τ. For every protocol P, we define its *set of traces* as follows:

$$\mathsf{trace}(P) = \{(\mathsf{tr}, \phi) \mid P \overset{\mathsf{tr}}{\Longrightarrow} (\mathcal{P}; \phi) \text{ for some configuration } (\mathcal{P}; \phi)\}.$$

Note that, by definition of $\mathsf{trace}(P)$, $\mathsf{tr}\phi\downarrow$ only contains terms from $\mathcal{T}_0(\Sigma_\mathsf{c}, \Sigma_0 \cup \mathcal{N})$.

Example 3. Consider the following sequence tr:

$$\begin{aligned}\mathsf{tr} = \mathsf{out}(c_A, ch_1).\mathsf{out}(c_B, ch_2).\mathsf{out}(ch_1, \mathsf{w}_1).\mathsf{in}(ch_2, \mathsf{w}_1).\\ \mathsf{out}(ch_2, \mathsf{w}_2).\mathsf{in}(ch_2, R_0).\mathsf{out}(ch_2, \mathsf{w}_3).\mathsf{in}(ch_1, \mathsf{w}_3)\end{aligned}$$

where $R_0 = \langle \mathsf{proj}_{1/5}(\mathsf{w}_2), \mathsf{proj}_{4/5}(\mathsf{w}_2), \mathsf{proj}_{5/5}(\mathsf{w}_2)\rangle$, and $\mathsf{proj}_{i/5}$ is used as a shortcut to extract the i^{th} component of a 5-uplet. Actually such a sequence of actions allows one to reach the following frame with $t_{\mathsf{enc}} = \mathsf{enc}(\langle n_a, m, \mathsf{a}, \mathsf{b}\rangle, k_{as})$:

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

We have that $(\mathsf{tr}, \phi) \in \mathsf{trace}(P_{\mathsf{OR}})$. The first five actions actually correspond to a normal execution of the protocol. Then, the agent who plays P_B will accept in input the message built using R_0, *i.e.* $u = \langle m, \mathsf{enc}(\langle n_a, m, \mathsf{a}, \mathsf{b}\rangle, k_{as}), \mathsf{enc}(\langle n_b, m, \mathsf{a}, \mathsf{b}\rangle, k_{bs})\rangle$. Indeed, this message has the expected form. At this stage, the agent who plays P_B is waiting for a message of the form: $u_0 = \langle m, z_{as}, \mathsf{enc}(\langle n_b, y_{ab}\rangle, k_{bs})\rangle$. The substitution $\sigma = \{z_{as} \triangleright t_{\mathsf{enc}},\ y_{ab} \triangleright \langle m, \mathsf{a}, \mathsf{b}\rangle\}$ is such that $u = u_0\sigma$. Once this input has been done, a message is outputted (action $\mathsf{out}(ch_3, \mathsf{w}_3)$) and given in input to P_A (action $\mathsf{in}(ch_1, \mathsf{w}_3)$).

Note that, at the end of the execution, A and B share a key but it is not the expected one, *i.e.* one freshly generated by the trusted server, but $\langle m, \mathsf{a}, \mathsf{b}\rangle$.

2.3 Trace equivalence

Intuitively, two protocols are equivalent if they cannot be distinguished by any attacker. Trace equivalence can be used to formalise many interesting security properties, in particular privacy-type properties, such as those studied for instance in [9]. We first introduce a notion of intruder's knowledge well-suited to cryptographic primitives for which the success of decrypting is visible.

Definition 2. *Two frames ϕ_1 and ϕ_2 are* statically equivalent*, $\phi_1 \sim \phi_2$, when we have that $dom(\phi_1) = dom(\phi_2)$, and:*

- *for any recipe R, $R\phi_1{\downarrow} \in \mathcal{T}_0(\Sigma_{\mathsf{c}}, \Sigma_0 \cup \mathcal{N})$ iff $R\phi_2{\downarrow} \in \mathcal{T}_0(\Sigma_{\mathsf{c}}, \Sigma_0 \cup \mathcal{N})$; and*
- *for all recipes R_1 and R_2 such that $R_1\phi_1{\downarrow}, R_2\phi_1{\downarrow} \in \mathcal{T}_0(\Sigma_{\mathsf{c}}, \Sigma_0 \cup \mathcal{N})$, we have that $R_1\phi_1{\downarrow} = R_2\phi_1{\downarrow}$ iff $R_1\phi_2{\downarrow} = R_2\phi_2{\downarrow}$.*

Intuitively, two frames are equivalent if an attacker cannot see the difference between the two situations they represent. If some computation fails in ϕ_1 for some recipe R, *i.e.* $R\phi_1{\downarrow}$ is not a message, it should fail in ϕ_2 as well. Moreover, ϕ_1 and ϕ_2 should satisfy the same equalities. In other words, the ability of the attacker to distinguish whether a recipe R produces a message, or whether two recipes R_1, R_2 produce the same message should not depend on the frame.

Example 4. Consider $\phi_1 = \phi \cup \{\mathsf{w}_4 \triangleright \langle m, \mathsf{a}, \mathsf{b}\rangle\}$, and $\phi_2 = \phi \cup \{\mathsf{w}_4 \triangleright n\}$ where n is a name. Let $R = \mathsf{proj}_1(\mathsf{w}_4)$. We have that $R\phi_1{\downarrow} = m \in \mathcal{T}_0(\Sigma_c, \Sigma_0 \cup \mathcal{N})$, but $R\phi_2{\downarrow} = \mathsf{proj}_1(n) \notin \mathcal{T}_0(\Sigma_c, \Sigma_0 \cup \mathcal{N})$, hence $\phi_1 \not\sim \phi_2$. This non static equivalence can also be established considering the recipes $R_1 = \langle \mathsf{proj}_1(\mathsf{w}_3), \mathsf{a}, \mathsf{b}\rangle$ and $R_2 = \mathsf{w}_4$. We have that $R_1\phi_1{\downarrow}, R_2\phi_1{\downarrow} \in \mathcal{T}_0(\Sigma_{\mathsf{c}}, \Sigma_0 \cup \mathcal{N})$, and $R_1\phi_1{\downarrow} = R_2\phi_1{\downarrow}$ whereas $R_1\phi_2{\downarrow} \neq R_2\phi_2{\downarrow}$.

Intuitively, two protocols are *trace equivalent* if, however they behave, the resulting sequences of messages observed by the attacker are in static equivalence.

Definition 3. *A protocol P is* trace included *in a protocol Q, written $P \sqsubseteq Q$, if for every $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$, there exists $(\mathsf{tr}', \phi') \in \mathsf{trace}(Q)$ such that $\mathsf{tr} = \mathsf{tr}'$ and $\phi \sim \phi'$. The protocols P and Q are* trace equivalent*, written $P \approx Q$, if $P \sqsubseteq Q$ and $Q \sqsubseteq P$.*

As illustrated by the following example, restricting messages to only contain atoms in key position also provides the adversary with more comparison power when variables occurred in key position in the protocol.

Example 5. Let $n, k \in \mathcal{N}$ and consider the protocol $P = \mathsf{in}(c, x).\mathsf{out}(c, \mathsf{enc}(n, k))$ as well as the protocol $Q = \mathsf{in}(c, x).\mathsf{out}(c, \mathsf{enc}(\mathsf{enc}(n, x), k))$. An attacker may distinguish between P and Q by sending a non atomic data and observing whether the process can emit. Q will not be able to emit since its first encryption will fail. This attack would not have been detected if arbitrary terms were allowed in key position.

In what follows, we consider *determinate* protocols as defined in [10], *i.e.*, we consider protocols in which the attacker knowledge is completely determined (up to static equivalence) by its past interaction with the protocol participants.

Definition 4. *A protocol P is* determinate *if for any tr, and for any $(\mathcal{P}_1, \phi_1)$, $(\mathcal{P}_2, \phi_2)$ such that $P \xRightarrow{\mathsf{tr}} (\mathcal{P}_1, \phi_1)$, and $P \xRightarrow{\mathsf{tr}} (\mathcal{P}_2, \phi_2)$, we have that $\phi_1 \sim \phi_2$.*

Assume given two determinate protocols P and Q such that $P \not\sqsubseteq Q$. A *witness of non-inclusion* is a trace tr for which there exists ϕ such that $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ and:

- either there does not exist ϕ' such that $(\mathsf{tr}, \phi') \in \mathsf{trace}(Q)$,
- or such a ϕ' exists and $\phi \not\sim \phi'$.

A *witness of non-equivalence* for determinate protocols P and Q is a trace tr that is a witness for $P \not\sqsubseteq Q$ or $Q \not\sqsubseteq P$. Note that when a protocol P is determinate, once the sequence tr is fixed, all the frames reachable through tr are actually in static equivalence, which ensures the unicity of ϕ', if it exists, up-to static equivalence.

Example 6. We wish to check strong secrecy of the exchanged key received by the agent A for the Otway-Rees protocol. A way of doing so is to check that $P^1_{\mathsf{OR}} \approx P^2_{\mathsf{OR}}$ where the two protocols are defined as follows:

- P^1_{OR} is as P_{OR} but we add the instruction $\mathsf{out}(c_1, x_{ab})$ at the end of the process P_A;
- P^2_{OR} is as P_{OR} but we add the instruction $\mathsf{new}\, n.\mathsf{out}(c_1, n)$ at the end of P_A.

The idea is to check whether an attacker can see the difference between the session key obtained by A and a fresh nonce.

As already suggested by the scenario described in Example 3, the secrecy (and so the strong secrecy) of the key received by A is not preserved. More precisely, consider the sequence $\mathsf{tr}' = \mathsf{tr}.\mathsf{out}(ch_1, \mathsf{w}_4)$ where tr is as in Example 3. In particular, $(\mathsf{tr}', \phi_1) \in \mathsf{trace}(P^1_{\mathsf{OR}})$ and $(\mathsf{tr}', \phi_2) \in \mathsf{trace}(P^2_{\mathsf{OR}})$ with $\phi_1 = \phi \cup \{\mathsf{w}_4 \triangleright \langle m, \mathsf{a}, \mathsf{b}\rangle\}$ and $\phi_2 = \phi \cup \{\mathsf{w}_4 \triangleright n\}$. As described in Example 4, $\phi_1 \not\sim \phi_2$ and thus tr' is a witness of non-equivalence for P^1_{OR} and P^2_{OR}. This witness is actually a variant of a known attack on the Otway-Rees protocol [15].

3 Existence of a well-typed witness of non-equivalence

In this section, we present our first main contribution: a simplification result that reduces the search space for attacks. Roughly, when looking for an attack, we can restrict ourselves to consider well-typed traces. This results holds for a general class of typing systems and as soon as the protocols under study are determinate and type-compliant. We first explain these hypotheses and then we state our general simplification result (see Theorem 1). The proof of this simplification result involves to provide a novel decision procedure for trace equivalence in the case of a bounded number of sessions. The novelty of this decision procedure, in comparison to the existing ones, is to provide a well-typed witness whenever the two processes are not in trace equivalence. This key intermediate result is stated in Proposition 1.

3.1 Typing system

Our simplification result holds for a general class of typing systems: we simply require that types are preserved by unification and application of substitutions. These operations are indeed routinely used in decision procedures.

Definition 5. *A* typing system *is a pair $(\mathcal{T}, \delta)$ where $\mathcal{T}$ is a set of elements called* types*, and δ is a function mapping terms $t \in \mathcal{T}(\Sigma_{\mathsf{c}}, \Sigma_0 \cup \mathcal{N} \cup \mathcal{X})$ to types τ in $\mathcal{T}$ such that:*

- *if t is a term of type τ and σ is a* well-typed *substitution,* i.e. *every variable of its domain has the same type as its image, then $t\sigma$ is of type τ,*

– *for any terms t and t' with the same type,* i.e. $\delta(t) = \delta(t')$ *and which are unifiable, their most general unifier ($mgu(t, t')$) is well-typed.*

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.

A straightforward typing system is when all terms are of a unique type, say Msg. Of course, our typing result would then be useless to reduce the search space for attacks. Which typing system shall be used typically depends on the protocols under study. We present in Section 5 a typing system that allows us to reduce the search space (and then derive decidability) for a large subclass of (tagged) protocols.

3.2 Well-typed trace

Whether or not a trace is well-typed is defined w.r.t. the set of *symbolic traces* of a protocol. Formally, we define $\xrightarrow{\mathsf{tr}_s}_s$ to be the transitive closure of the relation $\xrightarrow{\alpha_s}_s$ defined between processes as follows:

$$\mathsf{in}(c,u).P \cup \mathcal{P} \xrightarrow{\mathsf{in}(c,u)}_s P \cup \mathcal{P} \qquad !P \cup \mathcal{P} \xrightarrow{\tau}_s P' \cup !P \cup \mathcal{P}$$
$$\mathsf{out}(c,u).P \cup \mathcal{P} \xrightarrow{\mathsf{out}(c,u)}_s P \cup \mathcal{P} \qquad \mathsf{new}\, n.P \cup \mathcal{P} \xrightarrow{\tau}_s P\{^{n'}/_{n}\} \cup \mathcal{P}$$
$$\mathsf{new}\, c'.\mathsf{out}(c,c').P \cup \mathcal{P} \xrightarrow{\mathsf{out}(c,ch_i)}_s P\{^{ch_i}/_{c'}\} \cup \mathcal{P}$$

where P' is equal to P up to renaming of variables that do not occur yet in the trace with fresh ones (of the same type), n' is a fresh name (of the same type as n), and ch_i is the "next" fresh channel name available in $\mathcal{C}h^{\mathsf{fresh}}$.

Then, the set of *symbolic traces* $\mathsf{trace}_s(P)$ of a protocol P is defined as follows:

$$\mathsf{trace}_s(P) = \{\mathsf{tr}_s \mid P \xrightarrow{\mathsf{tr}_s}_s Q \text{ for some } Q\}.$$

Intuitively, the symbolic traces are simply all possible traces before instantiation of the variables, with some renaming to avoid unwanted captures.

Example 7. Let $P_1 = \mathsf{in}(c,x).!\mathsf{new}\, k.\, \mathsf{in}(c, \mathsf{enc}(\langle x, y\rangle, k))$. We have that:

$$\mathsf{tr}_s = \mathsf{in}(c,x).\mathsf{in}(c, \mathsf{enc}(\langle x, y_1\rangle, k_1)).\mathsf{in}(c, \mathsf{enc}(\langle x, y_2\rangle), k_2) \in \mathsf{trace}_s(P_1)$$

Indeed, the variable x is bound before replication.

As stated in the lemma below, any concrete trace is the instance of a symbolic trace.

Lemma 1. *Let P be a protocol and $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$. We have that $\mathsf{tr}\phi{\downarrow} = \mathsf{tr}_s\sigma$ for some $\mathsf{tr}_s \in \mathsf{trace}_s(P)$ and some substitution σ.*

A well-typed trace is simply a trace that is well-typed w.r.t. one of the symbolic traces. Since keys are atomic, some executions may fail when a protocol is about to output a term that contains an encryption with a non atomic key. To detect these behaviours, we need to consider slightly ill-typed traces. Formally, we consider a special constant $\omega \in \Sigma_0$. Its usefulness is illustrated in Example 8.

Definition 6. *A* first-order trace *of P is a sequence $\mathsf{tr} = \mathsf{tr}_s\sigma$ where $\mathsf{tr}_s \in \mathsf{trace}_s(P)$ and σ is a substitution such that for any $\mathsf{io}(c,u)$ that occurs in tr_s with $\mathsf{io} \in \{\mathsf{in}, \mathsf{out}\}$ and u not a channel, then $u\sigma \in \mathcal{T}_0(\Sigma_c, \Sigma_0 \cup \mathcal{N} \cup \mathcal{X})$. The trace tr is said to be:*

– well-typed *w.r.t. a typing system* $(\mathcal{T}, \delta)$ *if there exists such a* σ *that is well-typed;*
– pseudo-well-typed *w.r.t. a typing system* $(\mathcal{T}, \delta)$ *if there exists such* σ*, as well as* $\mathsf{c}_0 \in \Sigma_0$ *and* σ' *such that* $\sigma = \sigma'\{^{\langle \omega, \omega \rangle}/_{\mathsf{c}_0}\}$ *with* σ' *well-typed.*

Then a trace $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ *is* well-typed *(resp.* pseudo-well-typed*) if* $\mathsf{tr}\phi\!\downarrow$ *is well-typed (resp.* pseudo-well-typed*).*

Note that Lemma 1 ensures that $\mathsf{tr}\phi\!\downarrow$ is a first-order trace of P, and a well-typed trace is also pseudo-well-typed.

Example 8. Going back to Example 5, let $\mathsf{tr} = \mathsf{in}(c, \langle \omega, \omega \rangle).\mathsf{out}(c, \mathsf{w}_1)$. We have that $(\mathsf{tr}, \{\mathsf{w}_1 \triangleright \mathsf{enc}(n, k)\}) \in \mathsf{trace}(P)$ while there exists no frame ψ such that $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$. Consider the typing system $(\mathcal{T}, \delta)$ such that $\delta(t) = \mathsf{atom}$ for any atom or variable t and $\delta(t) = \neg\mathsf{atom}$ if t is not an atom. We can see there exists no well-typed witness of $P \not\approx Q$ (while P and Q are type-compliant as defined in Definition 7). However, the witness $(\mathsf{tr}, \{\mathsf{w}_1 \triangleright \mathsf{enc}(n, k)\})$ of $P \not\sqsubseteq Q$ is pseudo-well-typed (note that $\langle \omega, \omega \rangle$ occurs in tr). Intuitively, pseudo-well-typed traces harness the ability for the attacker to use the protocol as an oracle to test if some terms (when used in a key position) are atomic.

3.3 Type compliance

Our main assumption on the typing of protocols is that any two unifiable encrypted subterms are of the same type. The goal of this part is to state this hypothesis formally.

Due to the presence of replication, we need to consider two copies of protocols in order to consider different instances of the variables. Given a protocol P with replication, we define its 2-unfolding $\mathsf{unfold}^2(P)$ to be the protocol such that every occurrence of a process $!R$ in P is replaced by $R \mid R$, and some α-renaming is performed on one copy to ensure names and variables distinctness of the resulting process. Note that if P is a protocol that does not contain any replication, we have that $\mathsf{unfold}^2(P) = P$.

Example 9. Let P_1 be the protocol defined in Example 7. We have that:
$\mathsf{unfold}^2(P_1) = \mathsf{in}(c, x).(\mathsf{new}\, k_1.\mathsf{in}(c, \mathsf{enc}(\langle x, y_1 \rangle, k_1)) \mid \mathsf{new}\, k_2.\mathsf{in}(c, \mathsf{enc}(\langle x, y_2 \rangle, k_2)))$

We write $St(t)$ for the set of *(syntactic) subterms* of a term t, and $ESt(t)$ the set of its *encrypted subterms*, *i.e.* $ESt(t) = \{u \in St(t) \mid u \text{ is of the form } \mathsf{enc}(u_1, u_2)\}$. We extend this notion to sets/sequences of terms, and to protocols as expected.

Definition 7. *A protocol* P *is* type-compliant *w.r.t. a typing system* $(\mathcal{T}, \delta)$ *if for every* $t, t' \in ESt(\mathsf{unfold}^2(P))$ *we have that:* t *and* t' *unifiable implies that* $\delta(t) = \delta(t')$.

3.4 Main result

We are now ready to state our first main contribution: if there is an attack, then there is a pseudo-well-typed attack. This result holds for protocols with replications and nonces.

Theorem 1. *Let* P *and* Q *be two determinate protocols type-compliant w.r.t.* $(\mathcal{T}_1, \delta_1)$ *and* $(\mathcal{T}_2, \delta_2)$ *respectively. We have that* $P \not\approx Q$ *if, and only if, there exists a witness of non-equivalence* tr *such that:*

– *either* $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ *for some* ϕ *and* (tr, ϕ) *is pseudo-well-typed w.r.t.* $(\mathcal{T}_1, \delta_1)$*;*
– *or* $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$ *for some* ψ *and* (tr, ψ) *is pseudo-well-typed w.r.t.* $(\mathcal{T}_2, \delta_2)$.

The key step for proving Theorem 1 is to provide a decision procedure, in the bounded case (*i.e.* processes without replication), that returns a pseudo-well-typed witness of non-equivalence.

Proposition 1. *Let P and Q be two determinate protocols without replication. There exists an algorithm that decides whether $P \approx Q$ and if not, returns a witness* tr *of non-equivalence. Moreover, if P and Q are type-compliant w.r.t. $(\mathcal{T}_1, \delta_1)$ and $(\mathcal{T}_2, \delta_2)$ respectively, the witness* tr *of non-equivalence returned by the algorithm is such that:*

– *either* $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ *for some* ϕ *and* (tr, ϕ) *is pseudo-well-typed w.r.t.* $(\mathcal{T}_1, \delta_1)$*;*
– *or* $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$ *for some* ψ *and* (tr, ψ) *is pseudo-well-typed w.r.t.* $(\mathcal{T}_2, \delta_2)$.

The main idea is to assume given a decision procedure (for a bounded number of sessions) for reachability properties such as those proposed in [20, 16, 23] and to built on top of it a decision procedure for trace equivalence. Our procedure is carefully design to only allow unification between encrypted subterms. To achieve this,

1. we use as a reachability blackbox one that satisfies this requirement. Most of the existing algorithms (*e.g.* [20, 16, 23]) were not designed with such a goal in mind. However, in the case of the algorithm given in [16], it has already been shown how it can be turned into one that satisfies this requirement [17].
2. we design carefully the remaining of our algorithm to only consider unification between encrypted subterms.

This design allows us to provide a pseudo-well-typed witness when the protocols under study are type-compliant and not trace equivalent.

Then, relying on Proposition 1, the proof of Theorem 1 is almost immediate. Indeed, whenever two determinate type-compliant protocols P and Q are not in trace equivalence, there exists a witness of non-inclusion for $P \sqsubseteq Q$ (or $Q \sqsubseteq P$) for a bounded version of P and Q (unfolding the replications).

4 Decidability result

Now, assuming finitely many terms of each type, and in particular finitely many nonces, we obtain a new decidability result for trace equivalence, for an unbounded number of sessions. Compared to [13], we no longer need to restrict the number of variables per transition (to one), we allow variables in key positions, and we are more flexible in the control-flow of the program (we may have arbitrary sequences of in and out actions).

4.1 Simple processes

To establish decidability, we consider the class of simple protocols as given in [12] but we do not allow name restriction. Intuitively, simple protocols are protocols such that each copy of a replicated process has its own channel. This reflects the fact that due to IP addresses and sessions identifiers, an attacker can identify which process and which session he is sending messages to (or receiving messages from).

Definition 8. *A* simple protocol P *is a protocol of the form* $P_U \mid P_B$ *where:*

- $P_U = !\mathsf{new}\ c'_1.\mathsf{out}(c_1, c'_1).B_1 \mid \ldots \mid !\mathsf{new}\ c'_m.\mathsf{out}(c_m, c'_m).B_m$*; and*
- $P_B = B_{m+1} \mid \ldots \mid B_{m+n}$.

Each B_i *with* $1 \leq i \leq m$ *(resp.* $m < i \leq m+n$*) is a ground process on channel* c'_i *(resp.* c_i*) built using the following grammar:*

$$B := 0 \mid \mathsf{in}(c'_i, u).B \mid \mathsf{out}(c'_i, u).B \text{ where } u \in \mathcal{T}_0(\Sigma_\mathsf{c}, \Sigma_0 \cup \mathcal{N} \cup \mathcal{X}).$$

Moreover, we assume that $c_1, \ldots, c_n, c_{n+1}, \ldots, c_{n+m}$ *are pairwise distinct.*

Example 10. The protocol presented in Example 2 is not simple yet: we need to consider only finitely many nonces. To achieve this, we may remove all the instructions "$\mathsf{new}\ n$" with $n \in \mathcal{N}$ that occur in the process. Note that removing for instance "$\mathsf{new}\ n_a$" from the process P_A means that n_a is still modelled as a name, and thus it is unknown to the attacker. However, we do not assume anymore that a fresh nonce is generated at each session.

Simple protocols form a large class of protocols that are determinate: the attacker knows exactly who is sending a message or from whom he is receiving a message. Given a simple protocol P and a sequence of observable actions tr, there is a unique configuration $(\mathcal{P}; \phi)$ (up to some internal reduction steps) such that $P \stackrel{\mathsf{tr}}{\Longrightarrow} (\mathcal{P}; \phi)$.

Lemma 2. *A simple protocol is determinate.*

4.2 Main result

Our decidability result relies on the assumption that there are finitely many terms of each type (of the protocol), once the number of constants is bound for each type.

Formally, we say that a typing system $(\mathcal{T}, \delta)$ is *finite* if, for any set $\mathsf{A} \subseteq \mathcal{N} \cup \Sigma_0$ such that there is a finite number of names/constants of each type, then there are finitely many terms of each type, that is, for any $\tau \in \mathcal{T}$, the following set is finite and computable:

$$\{t \in \mathcal{T}(\Sigma_\mathsf{c}, \mathsf{A}) \mid \delta(t) = \tau\}.$$

Theorem 2. *The problem of deciding whether two simple protocols* P *and* Q*, type-compliant w.r.t. some finite typing systems* $(\mathcal{T}_1, \delta_1)$ *and* $(\mathcal{T}_2, \delta_2)$ *are trace equivalent (*i.e. $P \approx Q$*) is decidable.*

Proof. (Sketch) Since simple protocols are determinate (see Lemma 2), we obtain, thanks to our typing result (Theorem 1), the existence of well-typed witness of non-equivalence when such a witness exists. We further show that we can bound the number of useful constants in the witness trace. We then derive from the finiteness of the typing system that the witness trace uses finitely many distinct terms. Therefore, after some point, the trace only reproduces already existing transitions. Using the form of simple protocols, we can then show how to shorten the length of the witness trace. □

5 Application: tagged protocols

In this section, we instantiate our general results (Theorems 1 and 2) by exhibiting a class of protocols that is type-compliant for rather fine-grained typing systems. We consider tagged protocols, for a notion of tagging similar to one introduced by Blanchet [8].

Assume given a protocol P and an unfolding P' of it (remember that when computing $\mathsf{unfold}^2(P)$ names and variables are renamed to avoid clashes). Let u be a term in $\mathcal{T}(\Sigma_c, \Sigma_P \cup \mathcal{N}'_P \cup \mathcal{X}'_P)$ where Σ_P, $\mathcal{N}'_P$, $\mathcal{X}'_P$ are the constants, names, and variables occurring in P', we denote by $\overline{u}$ the transformation that replaces any name and variable occurring in u by its representative in $\mathcal{N}_P$ and $\mathcal{X}_P$ where $\mathcal{N}_P$ and $\mathcal{X}_P$ are the names and variables occurring in P.

Definition 9. *A protocol P is* tagged *if there exists a substitution σ_P such that for any $s_1, s_2 \in ESt(\mathsf{unfold}^2(P))$ with s_1 and s_2 unifiable, we have that $\overline{s_1\sigma_P} = \overline{s_2\sigma_P}$.*

Tagging can easily be enforced by labelling encrypted terms, as proposed in [8].

Definition 10. *A protocol P is* strongly tagged *if:*

1. *any term in $ESt(P)$ is of the form $\mathsf{enc}(\langle \mathsf{c}, m\rangle, k)$ for some $\mathsf{c} \in \Sigma_0$; and*
2. *there exists σ_P such that for any $s, t \in ESt(P)$ with $s = \mathsf{enc}(\langle \mathsf{c}_0, s_1\rangle, s_2)$ and $t = \mathsf{enc}(\langle \mathsf{c}_0, t_1\rangle, t_2)$ for some $\mathsf{c}_0 \in \Sigma_0$, we have that $s\sigma_P = t\sigma_P$.*

The second condition requires that there is a a substitution that unifies any two tagged terms unless their tags differ. This condition is easy to achieve for executable protocols. More precisely, assume a protocol admits an execution where each protocol step (in and out) is executed once (*i.e.* there is one honest execution). This protocol can be easily strongly tagged by adding a distinct tag in each encrypted term.

Lemma 3. *Let P be a protocol. If P is strongly tagged then P is tagged.*

Example 11. In our modelling of the Otway-Rees protocol, the protocols P^1_{OR} and P^2_{OR} (as described in Example 6) are not tagged. For instance, consider the terms $s_1 = \mathsf{enc}(\langle n_a, m, \mathsf{a}, \mathsf{b}\rangle, k_{as})$ and $s_2 = \mathsf{enc}(\langle n_a, x_{ab}\rangle, k_{as})$. Both are encrypted subterms of P_A (and thus of $\mathsf{unfold}^2(P^1_{\mathsf{OR}})$ and $\mathsf{unfold}^2(P^2_{\mathsf{OR}})$) and s_1 and s_2 are unifiable. Now, let $s_3 = \mathsf{enc}(\langle z_a, k_{ab}\rangle, k_{as})$. Actually, s_3 is an encrypted subterm of P_S which is unifiable with s_2. However, there exists no substitution σ such that $\overline{s_1\sigma} = \overline{s_2\sigma} = \overline{s_3\sigma}$.

We can consider a tagged, and safer, version of the Otway-Rees protocol by introducing 4 different tags, denoted 1,2,3 and 4, that are modelled using constants from Σ_0.

$$P'_{\mathsf{OR}} = !\,\mathsf{new}\ c_1.\mathsf{out}(c_A, c_1).P'_A \mid !\,\mathsf{new}\ c_2.\mathsf{out}(c_B, c_2).P'_B \mid !\,\mathsf{new}\ c_3.\mathsf{out}(c_S, c_3).P'_S$$

$$\begin{aligned}
P'_A &= \mathsf{new}\ m.\mathsf{new}\ n_a.\,\mathsf{out}(c_1, \langle m, \mathsf{a}, \mathsf{b}, \mathsf{enc}(\langle 1, n_a, m, \mathsf{a}, \mathsf{b}\rangle, k_{as})\rangle).\\
&\quad\ \mathsf{in}(c_1, \langle m, \mathsf{enc}(\langle 2, n_a, x_{ab}\rangle, k_{as})\rangle)\\
P'_B &= \mathsf{in}(c_2, \langle y_m, \mathsf{a}, \mathsf{b}, y_{as}\rangle).\\
&\quad\ \mathsf{new}\ n_b.\,\mathsf{out}(c_2, \langle y_m, \mathsf{a}, \mathsf{b}, y_{as}, \mathsf{enc}(\langle 3, n_b, y_m, \mathsf{a}, \mathsf{b}\rangle, k_{bs})\rangle).\\
&\quad\ \mathsf{in}(c_2, \langle y_m, z_{as}, \mathsf{enc}(\langle 4, n_b, y_{ab}\rangle, k_{bs})\rangle).\mathsf{out}(c_2, \langle y_m, z_{as}\rangle)\\
P'_S &= \mathsf{in}(c_3, \langle z_m, \mathsf{a}, \mathsf{b}, \mathsf{enc}(\langle 1, z_a, z_m, \mathsf{a}, \mathsf{b}\rangle, k_{as}), \mathsf{enc}(\langle 3, z_b, z_m, \mathsf{a}, \mathsf{b}\rangle, k_{bs})\rangle).\\
&\quad\ \mathsf{new}\ k_{ab}.\,\mathsf{out}(c_3, \langle z_m, \mathsf{enc}(\langle 2, z_a, k_{ab}\rangle, k_{as}), \mathsf{enc}(\langle 4, z_b, k_{ab}\rangle, k_{bs})\rangle)
\end{aligned}$$

and P'^1_{OR} and P'^2_{OR} are defined similarly as P^1_{OR} and P^2_{OR} relying on P'_{OR} instead of P_{OR}. Note that tr' is no longer a witness of $P'^1_{\mathsf{OR}} \not\approx P'^2_{\mathsf{OR}}$ as the attack has been removed by this tagging scheme. We can show that P'_{OR} is strongly tagged: consider the natural execution of P'_{OR}, matching inputs and outputs as intended. From this execution we can define:

$$\sigma_P = \{x_{ab} \triangleright k_{ab},\ y_m \triangleright m,\ y_{as} \triangleright \mathsf{enc}(\langle 1, n_a, m, \mathsf{a}, \mathsf{b}\rangle, k_{as})\rangle,\\ z_{as} \triangleright \mathsf{enc}(\langle 2, n_a, k_{ab}\rangle, k_{as}),\ z_m \triangleright m,\ z_a \triangleright n_a,\ z_b \triangleright n_b\}.$$

It is then easy to check that for any two terms s_1 and s_2 that are unifiable, their instances by σ_P are actually identical.

For any tagged protocol, we can infer a finite typing system, and show the type-compliance of the tagged protocol w.r.t. this typing system. Thus, relying on Theorem 2, we derive the following decidability result for simple and tagged protocols.

Corollary 1. *The problem of deciding whether two simple and tagged protocols P and Q are trace equivalent (*i.e. *$P \approx Q$) is decidable.*

Proof. (Sketch) The first step of the proof consists in associating to a tagged protocol P, a typing system $(\mathcal{T}_P, \delta_P)$ such that P is type-compliant w.r.t. $(\mathcal{T}_P, \delta_P)$. Intuitively, $(\mathcal{T}_P, \delta_P)$ is simply induced by σ_P, the substitution ensuring the tagged condition in Definition 9. For example, the type of a closed term t is t itself while the type of a variable x in P is simply $x\sigma_P$. This definition is then propagated to any term. With such typing systems, we can show that the size of a term (*i.e.* number of function symbols) is smaller than the size "indicated" by its type (*i.e.* the size of the type, viewed as a term). Thus the typing system $(\mathcal{T}_P, \delta_P)$ is finite. We then conclude by applying Theorem 2. □

Example 12. Consider the protocols $\overline{P'^1_{\mathsf{OR}}}$ and $\overline{P'^2_{\mathsf{OR}}}$ obtained from P'^1_{OR} and P'^2_{OR} by removing the instructions corresponding to a name restriction. These protocols are still strongly tagged and are now simple. Thus, our algorithm can be used to check whether these two protocols are in trace equivalence or not. This equivalence actually models a notion of strong secrecy of the key received by A. Since we have bounded the number of nonces, this equivalence does not require that the key is renewed at each session but it requires the key to be indistinguishable from a (private) name, n in our setting.

6 Conclusion

Decidability results for unbounded nonces are rare and complex, even in the reachability case. One of the only results has been established by Ramanujam and Suresh [21], assuming a particular tagging scheme (which itself involves nonces). We plan to explore whether our typing result could be applied to the tagging scheme defined in [21], to derive decidability of trace equivalence in the presence of nonces.

Our main typing result relies on the design of a new procedure in the case of a bounded number of sessions, that preserves typing. Specifically, we show that it is sufficient to consider only unification between encrypted (sub)terms. We think that this result can be applied to existing decision procedures (in particular SPEC [22] and also APTE [11], with some more work) to speed up their corresponding tools. As future work, we plan to implement this optimisation and measure its benefit.

References

1. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *28th Symposium on Principles of Programming Languages (POPL'01)*. ACM Press, 2001.
2. M. Abadi and R. M. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Trans. Software Eng.*, 22(1):6–15, 1996.
3. R. Amadio and W. Charatonik. On name generation and set-based analysis in the Dolev-Yao model. In *13th Int. Conference on Concurrency Theory (CONCUR'02)*, 2002.
4. M. Arapinis and M. Duflot. Bounding messages for free in security protocols. In *27th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'07)*, 2007.
5. M. Backes, C. Hritcu, and M. Maffei. Automated verification of remote electronic voting protocols in the applied pi-calculus. In *21st IEEE Computer Security Foundations Symposium (CSF'08)*, pages 195–209. IEEE Computer Society, 2008.
6. M. Baudet. Deciding security of protocols against off-line guessing attacks. In *12th ACM Conference on Computer and Communications Security (CCS'05)*. ACM Press, 2005.
7. B. Blanchet, M. Abadi, and C. Fournet. Automated Verification of Selected Equivalences for Security Protocols. In *20th Symposium on Logic in Computer Science*, 2005.
8. B. Blanchet and A. Podelski. Verification of cryptographic protocols: Tagging enforces termination. In *Foundations of Software Science and Computation Structures (FoSSaCS'03)*.
9. M. Bruso, K. Chatzikokolakis, and J. den Hartog. Formal verification of privacy for RFID systems. In *23rd Computer Security Foundations Symposium (CSF'10)*, 2010.
10. R. Chadha, Ş. Ciobâcă, and S. Kremer. Automated verification of equivalence properties of cryptographic protocols. In *21th European Symposium on Programming (ESOP'12)*, LNCS.
11. V. Cheval, H. Comon-Lundh, and S. Delaune. Trace equivalence decision: Negative tests and non-determinism. In *18th ACM Conference on Computer and Communications Security*.
12. V. Cheval, V. Cortier, and S. Delaune. Deciding equivalence-based properties using constraint solving. *Theoretical Computer Science*, 492:1–39, June 2013.
13. R. Chrétien, V. Cortier, and S. Delaune. From security protocols to pushdown automata. In *40th Int. Colloquium on Automata, Languages and Programming (ICALP'13)*, 2013.
14. R. Chrétien, V. Cortier, and S. Delaune. Typing messages for free in security protocols: the case of equivalence properties. Technical Report 8546, Inria, June 2014.
15. J. Clark and J. Jacob. A survey of authentication protocol literature: Version 1.0, 1997.
16. H. Comon-Lundh, V. Cortier, and E. Zalinescu. Deciding security properties for cryptographic protocols. Application to key cycles. *ACM Transactions on Computational Logic (TOCL)*, 11(4), 2010.
17. V. Cortier and S. Delaune. Safely composing security protocols. *Formal Methods in System Design*, 34(1):1–36, Feb. 2009.
18. N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*. Elsevier, 1990.
19. J. D. Guttman and F. J. Thayer. Protocol independence through disjoint encryption. In *13th Computer Security Foundations Workshop (CSFW'00)*. IEEE Comp. Soc. Press, 2000.
20. J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *8th ACM Conference on Computer and Communications Security*, 2001.
21. R. Ramanujam and S. P. Suresh. Tagging makes secrecy decidable with unbounded nonces as well. In *23rd Conference of Foundations of Software Technology and Theoretical Computer Science (FSTTCS'03)*, LNCS, pages 363–374. Springer, 2003.
22. A. Tiu and J. E. Dawson. Automating open bisimulation checking for the spi calculus. In *23rd IEEE Computer Security Foundations Symposium (CSF'10)*, pages 307–321, 2010.
23. A. Tiu, R. Goré, and J. E. Dawson. A proof theoretic analysis of intruder theories. *Logical Methods in Computer Science*, 6(3), 2010.