Bounding the number of agents, for equivalence too *

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

[1] LORIA, CNRS, France
[2] LSV, CNRS & ENS Cachan, Université Paris-Saclay, France

Abstract. Bounding the number of agents is a current practice when modeling a protocol. In 2003, it has been shown that one honest agent and one dishonest agent are indeed sufficient to find all possible attacks, for secrecy properties. This is no longer the case for equivalence properties, crucial to express many properties such as vote privacy or untraceability.
In this paper, we show that it is sufficient to consider two honest agents and two dishonest agents for equivalence properties, for deterministic processes with standard primitives and without else branches. More generally, we show how to bound the number of agents for arbitrary constructor theories and for protocols with simple else branches. We show that our hypotheses are tight, providing counter-examples for non action-deterministic processes, non constructor theories, or protocols with complex else branches.

1 Introduction

Many decision procedures and tools have been developed to automatically analyse cryptographic protocols. Prominent examples are ProVerif [8], Avispa [3], Scyther [18], or Tamarin [21], which have been successfully applied to various protocols of the literature. When modeling a protocol, it is common and necessary to make some simplifications. For example, it is common to consider a fix scenario with typically two honest and one dishonest agents. While bounding the number of sessions is known to be an unsound simplification (attacks may be missed), bounding the number of agents is a common practice which is typically not discussed. In 2003, it has been shown [15] that bounding the number of agents is actually a safe practice for trace properties. One honest agent and one dishonest agent are sufficient to discover all possible attacks against secrecy (for protocols without else branches). The reduction result actually holds for a large class of trace properties that encompasses authentication: if there is an attack then there is an attack with $b + 1$ agents where b is the number of agents used to *state* the security property.

* 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, the ANR project JCJC VIP n^o 11 JS02 006 01, and the DGA.

Trace properties are typically used to specify standard properties such as confidentiality or authentication properties. However, privacy properties such as vote privacy [19] or untraceability [2], or simply properties inherited from cryptographic games [16] (*e.g.* strong secrecy) are stated as equivalence properties. For example, Alice remains anonymous if an attacker cannot distinguish between a session with Alice from a session with Bob. When studying equivalence properties, the practice of bounding the number of agents has been continued. For example, most of the example files provided for equivalence in the ProVerif development [6] model only two or three agents.

The objective of this paper is to characterise when it is safe to bound the number of agents, for equivalence properties. In case of secrecy expressed as a trace property, bounding the number of agents is rather easy. If there is an attack then there is still an attack when projecting all honest agents on one single honest agent, and all dishonest agents on one single dishonest agent. This holds because the protocols considered in [15] do not have else branches: the conditionals only depend on equality tests that are preserved by projection.

Such a proof technique no longer works in case of equivalence. Indeed, an attack against an equivalence property may precisely rely on some disequality, which is not preserved when projecting several names on a single one. Consider for example a simple protocol where A authenticates to B by sending him her name, a fresh nonce, and a hash of these data.

$$A \to B : A, B, N, \mathsf{h}(A, N)$$

Let's denote this protocol by $P(A, B)$. This is clearly a wrong authentication protocol but let assume we wish to know whether it preserves A's privacy. In other words, is it possible for the attacker to learn whether A or A' is talking? That is, do we have $P(A, B)$ equivalent to $P(A', B)$? We need $A \neq A'$ to observe an attack, otherwise the two processes are identical. This example shows in particular that it is not possible to consider one honest agent and one dishonest agent as for trace properties.

Another issue comes from non deterministic behaviours. Non equivalence between P and Q is typically due to some execution that can be run in P and not in Q due to some failed test, that is, some disequality. Even if we maintain this disequality when projecting, maybe the projection enables new behaviours for Q, rendering it equivalent to P. Since non-determinism is usually an artefact of the modelling (in reality most protocols are perfectly deterministic), we assume in this paper *action-deterministic* protocols: the state of the system is entirely determined by the behaviour of the attacker. Such determinacy hypotheses already appear in several papers, in several variants [11, 10, 5].

Our contribution. We show that for equivalence, four agents are actually sufficient to detect attacks, for action-deterministic protocols without else branches and for the standard primitives. We actually provide a more general result, for arbitrary constructor theories and for protocols with (some) else branches. Equational theories are used to model cryptographic primitives, from standard ones (*e.g.* encryption, signature, or hash) to more subtle ones such as blind signa-

tures [19] or zero-knowledge proofs [4]. The notion of constructor theories (where agents can detect when decryption fails) has been introduced by B. Blanchet [7]. It captures many cryptographic primitives and in particular all the aforementioned ones, although associative and commutative properties (*e.g.* exclusive or) are out of their scopes since we assume the exchanged messages do not contain destructors. Else branches are often ignored when studying trace properties since most protocols typically abort when a test fails. However, a privacy breach may precisely come from the observation of a failure or from the observation of different error messages. A famous example is the attack found on the biometric French passport [12]. We therefore consider protocols with simple else branches, where error messages may be emitted in the else branches.

Our general reduction result is then as follows. We show that, for arbitrary constructor theories and action-deterministic protocols with simple else branches, we may safely bound the number of agents to $4b + 2$ where b is the *blocking factor* of the theory under consideration. Any theory has a (finite) blocking factor and the theories corresponding to standard primitives have a blocking factor of 1. Moreover, in case protocols do not have else branches, then the number of agents can be further reduced to $2b + 2$ ($b + 1$ honest agents and $b + 1$ dishonest agents), yielding a bound of 2 honest agents and 2 dishonest agents for protocols using standard primitives.

We show moreover that our hypotheses are tight. For example, and rather surprisingly, it is not possible to bound the number of agents with the pure equational theory $\mathsf{dec}(\mathsf{enc}(x, y), y)$ (assuming the function symbol dec may occur in messages as well). Similarly, we provide counter-examples when processes are not action-deterministic or when processes have non simple else branches.

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

Related work. Compared to the initial work of [15] for trace properties, we have considered the more complex case of equivalence properties. Moreover, we consider a more general framework with arbitrary constructor theories and protocols with (simple) else branches. Our proof technique is inspired from the proof of [14], where it is shown that if there is an attack against equivalence for arbitrary nonces, then there is still an attack for a fix number of nonces. Taking advantage of the fact that we bound the number of agents rather than the number of nonces, we significantly extend the result: (simple) else branches; general constructor theories with the introduction of the notion of b-blocking factor; general action-deterministic processes (instead of the particular class of simple protocols, which requires a particular structure of the processes); and protocols with phase (to model more game-based properties).

2 Model for security protocols

Security protocols are modelled through a process algebra inspired from the applied pi calculus [1]. Participants in a protocol are modelled as processes,

and the communication between them is modelled by means of the exchange of messages that are represented by terms.

2.1 Term algebra

We consider two infinite and disjoint sets of names: $\mathcal{N}$ is the set of *basic names*, which are used to represent keys, nonces, whereas $\mathcal{A}$ is the set of *agent names*, *i.e.* names which represent the agents identities. We consider two infinite and disjoint sets of variables, denoted $\mathcal{X}$ and $\mathcal{W}$. Variables in $\mathcal{X}$ typically refer to unknown parts of messages expected by participants while variables in $\mathcal{W}$ are used to store messages learnt by the attacker. Lastly, we consider two disjoint sets of *constant symbols*, denoted Σ_0 and Σ_{error}. Constants in Σ_0 will be used for instance to represent nonces drawn by the attacker and this set is assumed to be infinite, while constants in Σ_{error} will typically refer to error messages. We assume a *signature* Σ, *i.e.* a set of function symbols together with their arity. The elements of Σ are split into *constructor* and *destructor* symbols, *i.e.* $\Sigma = \Sigma_c \uplus \Sigma_d$. We denote $\Sigma^+ = \Sigma \uplus \Sigma_0 \uplus \Sigma_{\mathsf{error}}$, and $\Sigma_c^+ = \Sigma_c \uplus \Sigma_0 \uplus \Sigma_{\mathsf{error}}$.

Given a signature $\mathcal{F}$, and a set of atomic data A, we denote by $\mathcal{T}(\mathcal{F}, \mathsf{A})$ the set of *terms* built from atomic data A by applying function symbols in $\mathcal{F}$. Terms without variables are called *ground*. We denote by $\mathcal{T}(\Sigma_c^+, \mathcal{N} \cup \mathcal{A} \cup \mathcal{X})$ the set of *constructor terms*. The set of *messages* $\mathcal{M}_\Sigma$ is some subset of ground constructor terms. Given a set of atomic data A, an *A-renaming* is a function ρ such that $dom(\rho) \cup img(\rho) \subseteq A$. We assume $\mathcal{M}_\Sigma$ as well as $\mathcal{T}(\Sigma_c^+, \mathcal{N} \cup \mathcal{A} \cup \mathcal{X}) \backslash \mathcal{M}_\Sigma$ to be stable under any $\mathcal{A}$-renaming and $(\Sigma_0 \cup \Sigma_{\mathsf{error}})$-renaming. Intuitively, being a message or not should not depend on a particular constant or name.

Example 1. The standard primitives (symmetric and asymmetric encryption, signature, pair, and hash) are typically modelled by the following signature.

$$\Sigma_{\mathsf{std}} = \{\mathsf{enc}, \mathsf{dec}, \mathsf{shk_s}, \mathsf{aenc}, \mathsf{adec}, \mathsf{pub}, \mathsf{priv}, \mathsf{sign}, \mathsf{checksign}, \mathsf{h}, \langle\,\rangle, \mathsf{proj}_1, \mathsf{proj}_2, \mathsf{eq}\}.$$

The symbols enc and dec (resp. aenc and adec) of arity 2 represent symmetric (resp. asymmetric) encryption and decryption whereas $\mathsf{shk_s}, \mathsf{pub}, \mathsf{priv}$ are constructor keys of arity 1. Pairing is modelled using $\langle\,\rangle$ of arity 2, whereas projection functions are denoted proj_1 and proj_2 (both of arity 1). Signatures are represented by sign of arity 2 with an associated verification operator $\mathsf{checksign}$ of arity 3. Hash functions are modelled by h, of arity 1. Finally, we consider the function symbol eq to model equality test. This signature is split into two parts: we have that $\Sigma_c = \{\mathsf{enc}, \mathsf{aenc}, \mathsf{h}, \mathsf{sign}, \mathsf{shk_s}, \mathsf{pub}, \mathsf{priv}, \langle\,\rangle\}$ and $\Sigma_d = \Sigma_{\mathsf{std}} \setminus \Sigma_c$.

We denote $vars(u)$ the set of variables that occur in a term u. The application of a substitution σ to a term u is written $u\sigma$, and we denote $dom(\sigma)$ its *domain*. The *positions* of a term are defined as usual. The properties of cryptographic primitives are modelled through a rewriting system, *i.e.* a set of rewriting rules of the form $\mathsf{g}(t_1, \ldots, t_n) \to t$ where g is a destructor, and $t, t_1, \ldots, t_n$ are constructor terms. A term u can be rewritten in v if there is a position p in u, and a rewriting rule $\mathsf{g}(t_1, \ldots, t_n) \to t$ such that $u|_p = \mathsf{g}(t_1, \ldots, t_n)\theta$ for some substitution θ. Moreover, we assume that $t_1\theta, \ldots, t_n\theta$ as well as $t\theta$ are *messages*. We

only consider sets of rewriting rules that yield a convergent rewriting system. We denote $u{\downarrow}$ the *normal form* of a given term u.

A *constructor theory* $\mathcal{E}$ is given by a signature Σ together with a notion of messages $\mathcal{M}_\Sigma$, and a finite set of rewriting rules $\mathcal{R}$ (as decribed above) that defines a convergent rewriting system.

Example 2. The properties of the standard primitives are reflected through the theory $\mathcal{E}_{\mathsf{std}}$ induced by the following convergent rewriting system:

$$\mathsf{dec}(\mathsf{enc}(x,y),y) \to x \qquad \mathsf{proj}_i(\langle x_1,x_2\rangle) \to x_i \text{ with } i\in\{1,2\}.$$
$$\mathsf{adec}(\mathsf{aenc}(x,\mathsf{pub}(y)),\mathsf{priv}(y)) \to x \qquad \mathsf{checksign}(\mathsf{sign}(x,\mathsf{priv}(y)),x,\mathsf{pub}(y)) \to \mathsf{ok}$$
$$\mathsf{eq}(x,x) \to \mathsf{ok}$$

We may consider $\mathcal{M}_\Sigma$ to be $\mathcal{T}(\Sigma_c^+,\mathcal{N}\cup\mathcal{A})$ the set of all ground constructor terms. We may as well consider only terms with atomic keys for example.

Constructor theories are flexible enough to model all standard primitives. However, such a setting does not allow one to model for instance a decryption algorithm that never fails and always returns a message (*e.g.* $\mathsf{dec}(m,k)$).

For modelling purposes, we split the signature Σ into two parts, namely Σ_{pub} and Σ_{priv}, and we denote $\Sigma_{\mathsf{pub}}^+ = \Sigma_{\mathsf{pub}} \uplus \Sigma_0 \uplus \Sigma_{\mathsf{error}}$. An attacker builds his own messages by applying public function symbols to terms he already knows and that are available through variables in $\mathcal{W}$. Formally, a computation done by the attacker is a *recipe*, *i.e.* a term in $\mathcal{T}(\Sigma_{\mathsf{pub}}^+,\mathcal{W})$.

2.2 Process algebra

We assume an infinite set $\mathcal{C}h = \mathcal{C}h_0 \uplus \mathcal{C}h^{\mathsf{fresh}}$ of channels used to communicate, where $\mathcal{C}h_0$ and $\mathcal{C}h^{\mathsf{fresh}}$ are infinite and disjoint. Intuitively, channels of $\mathcal{C}h^{\mathsf{fresh}}$ are used to instantiate channels when they are generated during the execution of a protocol. They should not be part of a protocol specification. Protocols are modelled through processes using the following grammar:

$$\begin{array}{lll} P,Q = 0 & \mid \mathsf{let}\ x = v\ \mathsf{in}\ P\ \mathsf{else}\ 0 & \mid \mathsf{new}\, n.P \\ \quad \mid \mathsf{in}(c,u).P & \mid \mathsf{let}\ x = v\ \mathsf{in}\ P\ \mathsf{else}\ \mathsf{out}(c,\mathsf{err}) & \mid (P \mid Q) \\ \quad \mid \mathsf{out}(c,u).P & \mid\ !\,\mathsf{new}\, c'.\mathsf{out}(c,c').P & \mid i\colon P \end{array}$$

where $c,c' \in \mathcal{C}h$, $x \in \mathcal{X}$, $n \in \mathcal{N}$, $\mathsf{err} \in \Sigma_{\mathsf{error}}$, and $i \in \mathbb{N}$. We have that u is a constructor term, *i.e.* $u \in \mathcal{T}(\Sigma_c^+,\mathcal{N}\cup\mathcal{A}\cup\mathcal{X})$ whereas v can be any term in $\mathcal{T}(\Sigma^+,\mathcal{N}\cup\mathcal{A}\cup\mathcal{X})$.

Most of the constructions are rather standard. We may note the special construct $!\,\mathsf{new}\, c'.\mathsf{out}(c,c').P$ that combines replication with channel restriction. The goal of this construct, first introduced in [5], is to support replication while preserving some form of determinism, as formally defined later. Our calculus allows both message filtering in input actions as well as explicit application of destructor symbols through the let construction. The process "$\mathsf{let}\ x = v\ \mathsf{in}\ P\ \mathsf{else}\ Q$" tries to evaluate v and in case of success the process P is executed; otherwise the process is blocked or an error is emitted depending on what is indicated in Q. The

let instruction together with the eq theory introduced in Example 2 can encode the usual "if then else" construction. Indeed, the process if $u = v$ then P else Q can be written as $\mathsf{let}\ x = \mathsf{eq}(u, v)\ \mathsf{in}\ P\ \mathsf{else}\ Q$. Since P can be executed only if no destructor remains in the term $\mathsf{eq}(u, v)$, this implies that u and v must be equal. Our calculus also introduces a *phase* instruction, in the spirit of [9], denoted $i\colon P$. Some protocols like e-voting protocols may proceed in phase. More generally, phases are particularly useful to model security requirements, for example in case the attacker interacts with the protocol before being given some secret.

We denote by $fv(P)$ (resp. $fc(P)$) the set of free variables (resp. channels) that occur in a process P, *i.e.* those that are not in the scope of an input or a let construction (resp. new construction). A *basic process built on a channel* c is a process that contains neither | (parallel) nor ! (replication), and such that all its inputs/outputs take place on the channel c.

Example 3. The Denning Sacco protocol [20] is a key distribution protocol relying on symmetric encryption and a trusted server. It can be described informally as follows, in a version without timestamps:

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

where $\{m\}_k$ denotes the symmetric encryption of a message m with key k. Agent A (resp. B) communicates to a trusted server S, using a long term key K_{as} (resp. K_{bs}), shared with the server. At the end of a session, A and B should be authenticated and should share a session key K_{ab}.

We model the Denning Sacco protocol as follows. Let k be a name in $\mathcal{N}$, whereas a and b are names from $\mathcal{A}$. We denote by $\langle x_1, \ldots, x_{n-1}, x_n\rangle$ the term $\langle x_1, \langle \ldots \langle x_{n-1}, x_n\rangle\rangle\rangle$. The protocol is modelled by the parallel composition of three basic processes P_A, P_B, and P_S built respectively on c_1, c_2, and c_3. They correspond respectively to the roles of A, B, and S.

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

where processes P_A, P_B, and P_S are defined as follows.

- $P_A = \mathsf{out}(c_1, \langle a, b\rangle).\mathsf{in}(c_1, \mathsf{enc}(\langle b, x_{AB}, x_B\rangle, \mathsf{shk_s}(a))).\mathsf{out}(c_1, x_B)$
- $P_B = \mathsf{in}(c_2, \mathsf{enc}(\langle y_{AB}, a\rangle, \mathsf{shk_s}(b)))$
- $P_S = \mathsf{in}(c_3, \langle a, b\rangle).\ \mathsf{new}\ k.\mathsf{out}(c_3, \mathsf{enc}(\langle b, k, \mathsf{enc}(\langle k, a\rangle, \mathsf{shk_s}(b))\rangle, \mathsf{shk_s}(a))).$

2.3 Semantics

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

- $\mathcal{P}$ 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 $\mathcal{M}_\Sigma$.

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

$$\textsc{Const}\quad (i\colon \mathsf{out}(c,\mathsf{cst}).P \cup \mathcal{P};\ \phi;\ i) \xrightarrow{\mathsf{out}(c,\mathsf{cst})} (i\colon P \cup \mathcal{P};\ \phi;\ i) \quad \text{with } \mathsf{cst} \in \Sigma_0 \cup \Sigma_{\mathsf{error}}.$$

$$\textsc{Out}\quad (i\colon \mathsf{out}(c,u).P \cup \mathcal{P};\ \phi;\ i) \xrightarrow{\mathsf{out}(c,\mathsf{w})} (i\colon P \cup \mathcal{P};\ \phi \cup \{\mathsf{w} \triangleright u\};\ i) \quad \text{with } \mathsf{w} \text{ a fresh variable from } \mathcal{W}, \text{ and } u \in \mathcal{M}_\Sigma \setminus (\Sigma_0 \cup \Sigma_{\mathsf{error}}).$$

$$\textsc{Sess}\quad (i\colon !\,\mathsf{new}\,c'.\mathsf{out}(c,c').P \cup \mathcal{P};\ \phi;\ i) \xrightarrow{\mathsf{sess}(c,ch)} (i\colon P\{^{ch}/_{c'}\} \cup i\colon !\,\mathsf{new}\,c'.\mathsf{out}(c,c').P \cup \mathcal{P};\ \phi;\ i) \quad \text{with } ch \text{ a fresh name from } \mathcal{C}h^{\mathsf{fresh}}.$$

$$\textsc{Let}\quad (i\colon \mathsf{let}\,x = v\,\mathsf{in}\,P\,\mathsf{else}\,Q \cup \mathcal{P};\ \phi;\ i) \xrightarrow{\tau} (i\colon P\{^{v\downarrow}/_{x}\} \cup \mathcal{P};\ \phi;\ i) \quad \text{when } v\!\downarrow \in \mathcal{M}_\Sigma.$$

$$\textsc{Let-Fail}\quad (i\colon \mathsf{let}\,x = v\,\mathsf{in}\,P\,\mathsf{else}\,Q \cup \mathcal{P};\ \phi;\ i) \xrightarrow{\tau} (i\colon Q \cup \mathcal{P};\ \phi;\ i) \quad \text{when } v\!\downarrow \notin \mathcal{M}_\Sigma.$$

$$\textsc{Null}\quad (i\colon 0 \cup \mathcal{P};\ \phi;\ i) \xrightarrow{\tau} (\mathcal{P};\ \phi;\ i)$$

$$\textsc{Par}\quad (i\colon (P \mid Q) \cup \mathcal{P};\ \phi;\ i) \xrightarrow{\tau} (i\colon P \cup i\colon Q \cup \mathcal{P};\ \phi;\ i)$$

$$\textsc{New}\quad (i\colon \mathsf{new}\,n.P \cup \mathcal{P};\ \phi;\ i) \xrightarrow{\tau} (i\colon P\{^{n'}/_{n}\} \cup \mathcal{P};\ \phi;\ i) \quad \text{with } n' \text{ a fresh name from } \mathcal{N}.$$

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

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

$$\textsc{Clean}\quad (i\colon P \cup \mathcal{P};\ \phi;\ i') \xrightarrow{\tau} (\mathcal{P};\ \phi;\ i') \quad \text{when } i' > i.$$

Fig. 1. Semantics for processes

Intuitively, i is an integer that indicates the current phase; $\mathcal{P}$ represents the processes that still remain to be executed; and ϕ represents the sequence of messages that have been learnt so far by the attacker.

We often write P instead of $0\colon P$ or $(\{0\colon P\};\ \emptyset;\ 0)$. The operational semantics of a process P is induced by the relation $\xrightarrow{\alpha}$ over configurations as defined in Figure 1.

The rules are quite standard and correspond to the intuitive meaning of the syntax given in the previous section. When a process emits a message m, we distinguish the special case where m is a constant (Const rule), in which case the constant m appears directly in the trace instead of being stored in the frame. This has no impact on the intuitive behaviour of the process but is quite handy in the proofs. Regarding phases (rules Move, Phase, and Clean), the adversary may move to a subsequent phase whenever he wants while processes may move to the next phase when they are done or simply disappear if the phase is over.

Given a sequence of actions $\alpha_1 \dots \alpha_n$, the relation $\xrightarrow{\alpha_1 \dots \alpha_n}$ between configurations is defined as the transitive closure of $\xrightarrow{\alpha}$. Given a sequence of observable action tr, we denote $\mathcal{C} \overset{\mathsf{tr}}{\Rightarrow} \mathcal{C}'$ when there exists a sequence $\alpha_1, \dots, \alpha_n$ for some n such that $\mathcal{C} \xrightarrow{\alpha_1 \dots \alpha_n} \mathcal{C}'$, and tr is obtained from this sequence by removing all the unobservable τ actions.

Definition 1. *Given a configuration* $\mathcal{C} = (\mathcal{P}; \phi; i)$*, we denote* $\mathsf{trace}(\mathcal{C})$ *the set of traces defined as follows:*

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

Example 4. Let $\mathcal{C}_{\mathsf{DS}} = (P_{\mathsf{DS}}; \emptyset; 0)$ with P_{DS} as defined in Example 3. We have that $(\mathsf{tr}, \phi) \in \mathsf{trace}(\mathcal{C}_{\mathsf{DS}})$ where tr, and ϕ are as described below:

- $\mathsf{tr} = \mathsf{sess}(c_A, ch_1).\mathsf{sess}(c_B, ch_2).\mathsf{sess}(c_S, ch_3).\mathsf{out}(ch_1, \mathsf{w}_1).\mathsf{in}(ch_3, \mathsf{w}_1).$
  $\mathsf{out}(ch_3, \mathsf{w}_2).\mathsf{in}(ch_1, \mathsf{w}_2).\mathsf{out}(ch_1, \mathsf{w}_3).\mathsf{in}(ch_2, \mathsf{w}_3)$; and
- $\phi = \{\mathsf{w}_1 \triangleright \langle a, b\rangle,\ \mathsf{w}_2 \triangleright \mathsf{enc}(\langle b, k, \mathsf{enc}(\langle k, a\rangle, \mathsf{shk}_{\mathsf{s}}(b))\rangle, \mathsf{shk}_{\mathsf{s}}(a)),$
  $\mathsf{w}_3 \triangleright \mathsf{enc}(\langle k, a\rangle, \mathsf{shk}_{\mathsf{s}}(b))\}.$

This trace corresponds to a normal execution of the Denning Sacco protocol.

2.4 Action-determinism

As mentioned in introduction, we require processes to be deterministic. We provide in Section 4.3 an example showing why the number of agents may not be bound when processes are not deterministic. We consider a definition similar to the one introduced in [5], extended to process with phase.

Definition 2. *A configuration* $\mathcal{C}$ *is* action-deterministic *if whenever* $\mathcal{C} \xrightarrow{\mathsf{tr}} (\mathcal{P}; \phi; i)$*, and* $i{:}\,\alpha.P$ *and* $i{:}\,\beta.Q$ *are two elements of* $\mathcal{P}$ *with* α, β *instruction of the form* $\mathsf{in}(c, u)$*,* $\mathsf{out}(c, u)$ *or* $\mathsf{new}\, c'.\mathsf{out}(c, c')$ *then either the underlying channels* c *differ or the instructions are not of the same nature (that is,* α, β *are not both an input, nor both an output, nor both channel creations).*

A process P *is action-deterministic if* $\mathcal{C} = (P; \phi; 0)$ *is action-deterministic for any frame* ϕ*.*

For such protocols, the attacker knowledge is entirely determined (up to α-renaming) by its interaction with the protocol.

Lemma 1. *Let* $\mathcal{C}$ *be an action-deterministic configuration such that* $\mathcal{C} \overset{\mathsf{tr}}{\Rightarrow} \mathcal{C}_1$ *and* $\mathcal{C} \overset{\mathsf{tr}}{\Rightarrow} \mathcal{C}_2$ *for some* tr*,* $\mathcal{C}_1 = (\mathcal{P}_1; \phi_1; i_1)$*, and* $\mathcal{C}_2 = (\mathcal{P}_2; \phi_2; i_2)$*. We have that* $i_1 = i_2$*, and* ϕ_1 *and* ϕ_2 *are equal modulo* α*-renaming.*

2.5 Trace equivalence

Many privacy properties such as vote-privacy or untraceability are expressed as trace equivalence [19, 2]. Intuitively, two configurations are trace equivalent if an attacker cannot tell with which of the two configurations he is interacting. We first introduce a notion of equivalence between frames.

Definition 3. *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* $R \in \mathcal{T}(\Sigma^+_{\mathsf{pub}}, \mathcal{W})$*, we have that* $R\phi_1{\downarrow} \in \mathcal{M}_\Sigma$ *implies that* $R\phi_2{\downarrow} \in \mathcal{M}_\Sigma$*; and*

– *for any recipes* $R, R' \in \mathcal{T}(\Sigma^+_{\mathsf{pub}}, \mathcal{W})$ *such that* $R\phi_1\!\downarrow, R'\phi_1\!\downarrow \in \mathcal{M}_\Sigma$*, we have that:* $R\phi_1\!\downarrow = R'\phi_1\!\downarrow$ *implies* $R\phi_2\!\downarrow = R'\phi_2\!\downarrow$.

They are in static equivalence*, written* $\phi_1 \sim \phi_2$*, if* $\phi_1 \sqsubseteq_s \phi_2$ *and* $\phi_2 \sqsubseteq_s \phi_1$.

An attacker can see the difference between two sequences of messages if he is able to perform some computation that succeeds in ϕ_1 and fails in ϕ_2; or if he can build a test that leads to an equality in ϕ_1 and not in ϕ_2 (or conversely).

Example 5. Consider $\phi_1 = \phi \cup \{\mathsf{w}_4 \triangleright \mathsf{enc}(\mathsf{m}_1, k)\}$ and $\phi_2 = \phi \cup \{\mathsf{w}_4 \triangleright \mathsf{enc}(\mathsf{m}_2, k')\}$ where ϕ has been introduced in Example 4. The terms m_1, m_2 are public constants in Σ_0, and k' is a fresh name in $\mathcal{N}$. We have that the two frames ϕ_1 and ϕ_2 are statically equivalent. Intuitively, at the end of a normal execution between honest participants, an attacker can not make any distinction between a public constant m_1 encrypted with the session key, and another public contant m_2 encrypted with a fresh key k' that has never been used.

Trace equivalence is the active counterpart of static equivalence. Two configurations are trace equivalent if, however they behave, the resulting sequences of messages observed by the attacker are in static equivalence.

Definition 4. *Let* $\mathcal{C}$ *and* $\mathcal{C}'$ *be two configurations. They are in* trace equivalence*, written* $\mathcal{C} \approx \mathcal{C}'$*, if for every* $(\mathsf{tr}, \phi) \in \mathsf{trace}(\mathcal{C})$*, there exist* $(\mathsf{tr}', \phi') \in \mathsf{trace}(\mathcal{C}')$ *such that* $\mathsf{tr} = \mathsf{tr}'$*, and* $\phi \sim \phi'$ *(and conversely).*

Note that two trace equivalent configurations are necessary at the same phase. Of course, this is not a sufficient condition.

Example 6. The process P_{DS} presented in Example 3 models the Denning Sacco protocol. Strong secrecy of the session key, as received by the agent B, can be expressed by the following equivalence: $P^1_{\mathsf{DS}} \approx P^2_{\mathsf{DS}}$, where P^1_{DS} and P^2_{DS} are defined as follows. Process P^1_{DS} is process P_{DS} with the instruction $1\colon \mathsf{out}(c_2, \mathsf{enc}(\mathsf{m}_1, y_{AB}))$ added at the end of the process P_B; and P^2_{DS} is as the protocol P_{DS} with the instruction $1\colon \mathsf{new}\, k.\mathsf{out}(c_2, \mathsf{enc}(\mathsf{m}_2, k))$ at the end of P_B. The terms m_1 and m_2 are two public constants from Σ_0, and we use the phase instruction to make a separation between the protocol execution, and the part of the process that encodes the security property.

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

$$\mathsf{tr}' = \mathsf{tr}.\mathsf{sess}(c_B, ch_4).\mathsf{in}(ch_4, \mathsf{w}_3).\mathsf{phase}\ 1.\mathsf{out}(ch_2, \mathsf{w}_4).\mathsf{out}(ch_4, \mathsf{w}_5)$$

where tr has been defined in Example 4. The attacker simply replays an old session. The resulting (uniquely defined) frames are:

– $\phi'_1 = \phi \cup \{\mathsf{w}_4 \triangleright \mathsf{enc}(\mathsf{m}_1, k),\ \mathsf{w}_5 \triangleright \mathsf{enc}(\mathsf{m}_1, k)\}$; and
– $\phi'_2 = \phi \cup \{\mathsf{w}_4 \triangleright \mathsf{enc}(\mathsf{m}_2, k'),\ \mathsf{w}_5 \triangleright \mathsf{enc}(\mathsf{m}_2, k'')\}$.

Then $(\mathsf{tr}', \phi'_1) \in \mathsf{trace}(P^1_{\mathsf{DS}})$ and $(\mathsf{tr}', \phi'_2) \in \mathsf{trace}(P^2_{\mathsf{DS}})$. However, we have that $\phi'_1 \not\sim \phi'_2$ since $\mathsf{w}_4 = \mathsf{w}_5$ in ϕ'_1 but not in ϕ'_2. Thus P^1_{DS} and P^2_{DS} are *not* in trace equivalence. To avoid this attack, the original protocol relies on timestamps.

3 Results

Our main goal is to show that we can safely consider a bounded number of agents. Our result relies in particular on the fact that constructor theories enjoy the property of being *b-blockable*, which is defined in Section 3.2. Our main reduction result is then stated in Section 3.3 with a sketch of proof provided in Section 3.4. We first start this section with a presentation of our model for an unbounded number of agents.

3.1 Modelling an unbounded number of agents

In the previous section, for illustrative purposes, we considered a scenario that involved only 2 honest agents a and b. This is clearly not sufficient when performing a security analysis. To model an unbounded number of agents, we introduce some new function symbols $\Sigma_{\mathsf{ag}} = \{\mathsf{ag}, \mathsf{hon}, \mathsf{dis}\}$, each of arity 1. The term $\mathsf{ag}(a)$ with $a \in \mathcal{A}$ will represent the fact that a is an agent, $\mathsf{hon}(a)$ and $\mathsf{dis}(a)$ are intended to represent honest and compromised agents respectively. This distinction is used in protocol description to state the security property under study: typically, we wish to ensure security of data shared by *honest* agents. These symbols are private and not available to the attacker. We thus consider a term algebra as defined in Section 2. We simply assume in addition that $\Sigma_{\mathsf{ag}} \subseteq \Sigma_c \cap \Sigma_{\mathsf{priv}}$, and that our notion of messages contains at least $\{\mathsf{ag}(a), \mathsf{hon}(a), \mathsf{dis}(a) \mid a \in \mathcal{A}\}$.

Example 7. Going back to the Denning Sacco protocol presented in Example 3, we consider now a richer scenario.

$$P'_A = \mathsf{in}(c_1, \mathsf{ag}(z_A)).\mathsf{in}(c_1, \mathsf{ag}(z_B)).1\colon P_A$$
$$P'_B = \mathsf{in}(c_2, \mathsf{ag}(z_A)).\mathsf{in}(c_2, \mathsf{ag}(z_B)).1\colon P_B$$
$$P'_S = \mathsf{in}(c_3, \mathsf{ag}(z_A)).\mathsf{in}(c_3, \mathsf{ag}(z_B)).1\colon P_S$$

where P_A, P_B, and P_S are as defined in Example 3 after replacement of the occurrences of a (resp. b) by z_A (resp. z_B). Then the process P'_{DS} models an unbounded number of agents executing an unbounded number of sessions:

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

It is then necessary to provide an unbounded number of honest and dishonest agent names. This is the purpose of the following frame.

Definition 5. *Given an integer n, the frame $\phi_{\mathsf{hd}}(n) = \phi_{\mathsf{a}}(n) \uplus \phi_{\mathsf{h}}(n) \uplus \phi_{\mathsf{d}}(n)$ is defined as follows:*

- $\phi_{\mathsf{a}}(n) = \{\mathsf{w}_1^{\mathsf{h}} \triangleright a_1^h;\ \ldots;\ \mathsf{w}_n^{\mathsf{h}} \triangleright a_n^h;\ \mathsf{w}_1^{\mathsf{d}} \triangleright a_1^d;\ \ldots;\ \mathsf{w}_n^{\mathsf{d}} \triangleright a_n^d\}$;
- $\phi_{\mathsf{h}}(n) = \{\mathsf{w}_1^{\mathsf{hag}} \triangleright \mathsf{ag}(a_1^h);\ \mathsf{w}_1^{\mathsf{hon}} \triangleright \mathsf{hon}(a_1^h);\ \ldots;\ \mathsf{w}_n^{\mathsf{hag}} \triangleright \mathsf{ag}(a_n^h);\ \mathsf{w}_n^{\mathsf{hon}} \triangleright \mathsf{hon}(a_n^h)\}$;
- $\phi_{\mathsf{d}}(n) = \{\mathsf{w}_1^{\mathsf{dag}} \triangleright \mathsf{ag}(a_1^d);\ \mathsf{w}_1^{\mathsf{dis}} \triangleright \mathsf{dis}(a_1^d);\ \ldots;\ \mathsf{w}_n^{\mathsf{dag}} \triangleright \mathsf{ag}(a_n^d);\ \mathsf{w}_n^{\mathsf{dis}} \triangleright \mathsf{dis}(a_n^d)\}$;

where a_i^h, and a_i^d $(1 \leq i \leq n)$ are pairwise different names in $\mathcal{A}$.

Of course, to model faithfully compromised agents, it is important to reveal their keys to the attacker. This can be modelled through an additional process K that should be part of the initial configuration.

Example 8. Going back to our running example, we may disclose keys through the following process.

$$K = !\,\mathsf{new}\, c'.\mathsf{out}(c_K, c').\mathsf{in}(c', \mathsf{dis}(x)).\mathsf{out}(c', \mathsf{shk_s}(x)).$$

This process reveals all the keys shared between the server and a compromised agent. Strong secrecy of the exchanged key can be expressed by the following family of equivalences with $n \geq 0$:

$$\begin{array}{c}(P'_{\mathsf{DS}} \mid !\,\mathsf{new}\, c'_2.\mathsf{out}(c'_B, c'_2).P'_1 \mid K;\ \phi_{\mathsf{hd}}(n);\ 0)\\ \approx\\ (P'_{\mathsf{DS}} \mid !\,\mathsf{new}\, c'_2.\mathsf{out}(c'_B, c'_2).P'_2 \mid K;\ \phi_{\mathsf{hd}}(n);\ 0)\end{array}$$

where P'_1 and P'_2 are processes that are introduced to model our strong secrecy property as done in Example 6.

$$\begin{array}{ll}P'_1 = \mathsf{in}(c'_2, \mathsf{hon}(z_A)).\mathsf{in}(c'_2, \mathsf{hon}(z_B)). & P'_2 = \mathsf{in}(c'_2, \mathsf{hon}(z_A)).\mathsf{in}(c'_2, \mathsf{hon}(z_B)).\\ \quad 1{:}\,\mathsf{in}(c'_2, \mathsf{enc}(\langle y_{AB}, z_A\rangle, \mathsf{shk_s}(z_B))). & \quad 1{:}\,\mathsf{in}(c'_2, \mathsf{enc}(\langle y_{AB}, z_A\rangle, \mathsf{shk_s}(z_B))).\\ \quad 2{:}\,\mathsf{out}(c'_2, \mathsf{enc}(\mathsf{m}_1, y_{AB})) & \quad 2{:}\,\mathsf{new}\, k'.\mathsf{out}(c'_2, \mathsf{enc}(\mathsf{m}_2, k'))\end{array}$$

Our reduction result applies to a rather large class of processes. However, we have to ensure that their executions do not depend on specific agent names. Moreover, we consider processes with *simple* else branches: an else branche can only be the null process or the emission of an error message.

Definition 6. *A protocol P is a process such that $fv(P) = \emptyset$, and $fc(P) \cap \mathcal{C}h^{\mathsf{fresh}} = \emptyset$. We also assume that P does not use names in $\mathcal{A}$. Moreover, the constants from Σ_{error} only occur in the* else *part of a* let *instruction in P.*

Example 9. Considering $\Sigma_{\mathsf{error}} = \emptyset$, it is easy to see that the processes

$$P'_{\mathsf{DS}} \mid !\,\mathsf{new}\, c'_2.\mathsf{out}(c'_B, c'_2).P'_i \mid K$$

with $i \in \{1, 2\}$ are protocols. They only have trivial else branches.

3.2 Blocking equational theories

We aim at reducing the number of agents. To preserve equivalence, our reduction has to preserve equalities as well as disequalities. It also has to preserve the fact of being a message or not. We introduce the notion of *b-blockable* theories: a theory is b-blockable if it is always sufficient to leave b agents unchanged to preserve the fact of not being a message.

Definition 7. *A constructor theory $\mathcal{E}$ is* b-blockable *if for any term $t \in \mathcal{T}(\Sigma^+, \mathcal{N} \cup \mathcal{A}) \setminus \mathcal{M}_\Sigma$ in normal form, there exists a set of names $\mathsf{A} \subseteq \mathcal{A}$ of size at most b such that for any $\mathcal{A}$-renaming ρ with $(dom(\rho) \cup img(\rho)) \cap \mathsf{A} = \emptyset$, we have that $t\rho\downarrow \notin \mathcal{M}_\Sigma$.*

Example 10. Let $\mathsf{eq}_2 \in \Sigma_d$ be a symbol of arity 4, and $\mathsf{ok} \in \Sigma_c$ be a constant. Consider the two following rewriting rules:

$$\mathsf{eq}_2(x, x, y, z) \to \mathsf{ok} \quad \text{and} \quad \mathsf{eq}_2(x, y, z, z) \to \mathsf{ok}$$

This theory can be used to model disjonction. Intuitively, $\mathsf{eq}_2(u_1, u_2, u_3, u_4)$ can be reduced to ok when either $u_1 = u_2$ or $u_3 = u_4$. Note that this theory is *not* 1-blockable. Indeed, the term $t = \mathsf{eq}_2(a, b, c, d)$ is a witness showing that keeping one agent name unchanged is not sufficient to prevent the application of a rewriting rule on $t\rho$ (for any renaming ρ that leaves this name unchanged). Actually, we will show that this theory is 2-blockable.

A constructor theory is actually always b-blockable for some b.

Proposition 1. *Any constructor theory $\mathcal{E}$ is b-blockable for some $b \in \mathbb{N}$.*

We note $b(\mathcal{E})$ the *blocking factor* of $\mathcal{E}$. This is the smallest b such that the theory $\mathcal{E}$ is b-blockable. Actually, not only all the theories are b-blockable for some b, but this bound is quite small for most of the theories that are used to model cryptographic primitives.

Example 11. The theory $\mathcal{E}_{\mathsf{std}}$ given in Example 2 is 1-blockable whereas the theory given in Example 10 is 2-blockable. These results are an easy consequence of Lemma 2 stated below.

The blocking factor of a constructor theory is related to the size of critical tuples of the theory.

Definition 8. *A constructor theory $\mathcal{E}$ with a rewriting system $\mathcal{R}$ has a critical set of size k if there exist k distinct rules $\ell_1 \to r_1, \ldots, \ell_k \to r_k$ in $\mathcal{R}$, and a substitution σ such that $\ell_1\sigma = \ldots = \ell_k\sigma$.*

Lemma 2. *If a constructor theory $\mathcal{E}$ has no critical set of size $k+1$ with $k \geq 0$ then it is k-blockable.*

This lemma is a consequence of the proof of Proposition 1 (see [17]). From this lemma, we easily deduce that many theories used in practice to model security protocols are actually 1-blockable. This is the case of the theory $\mathcal{E}_{\mathsf{std}}$ and many variants of it. We may for instance add function symbols to model blind signatures, or zero-knowledge proofs.

3.3 Main result

We are now able to state our main reduction result.

Theorem 1. *Let P, Q be two action-deterministic protocols built on a constructor theory $\mathcal{E}$. If $(P;\, \phi_{\mathsf{hd}}(n_0);\, 0) \approx (Q;\, \phi_{\mathsf{hd}}(n_0);\, 0)$ where $n_0 = 2b(\mathcal{E}) + 1$ and $b(\mathcal{E})$ is the blocking factor of $\mathcal{E}$, we have that*

$$(P;\, \phi_{\mathsf{hd}}(n);\, 0) \approx (Q;\, \phi_{\mathsf{hd}}(n);\, 0) \text{ for any } n \geq 0.$$

Moreover, when P and Q have only let *construction with trivial else branches considering $n_0 = b(\mathcal{E}) + 1$ is sufficient.*

This theorem shows that whenever two protocols are not in trace equivalence, then they are already not in trace equivalence for a relatively small number of agents that does not depend on the protocols (but only on the underlying theory).

Example 12. Continuing our running example, thanks to Theorem 1, we only have to consider 4 agents (2 honest agents and 2 dishonest ones) for the theory $\mathcal{E}_{\mathsf{std}}$ introduced in Example 2, that corresponds to the standard primitives. Therefore we only have to perform the security analysis considering $\phi_{\mathsf{a}}(2) \uplus \phi_{\mathsf{h}}(2) \uplus \phi_{\mathsf{d}}(2)$ as initial frame.

This reduction result bounds a priori the number of agents involved in an attack. However, due to our setting, the resulting configurations are not written in their usual form (*e.g.* compromised keys are emitted through process K instead of being included in the initial frame). We show that it is possible to retrieve the equivalences written in a more usual form, after some clean-up transformations and some instantiations. This step is formalised in Proposition 2. We first define the notion of key generator process. The purpose of such a process is to provide long-term keys of compromised agents to the attacker.

Definition 9. *A key generator is an action-deterministic process K with no phase instruction in it. Moreover, for any $n \in \mathbb{N}$, we assume that there exists $\phi_K(n)$ with no occurrence of symbols in Σ_{ag}, and such that:*

- $\mathcal{C}_K^n = (K;\, \phi_{\mathsf{hd}}(n);\, 0) \overset{\mathsf{tr}}{\Rightarrow} (K';\, \phi_{\mathsf{hd}}(n) \uplus \phi_K(n);\, 0)$ *for some* tr *and* K'*;*
- $img(\phi) \subseteq img(\phi_K(n))$ *for any* $(\mathcal{P};\, \phi_{\mathsf{hd}}(n) \uplus \phi;\, 0)$ *reachable from* $\mathcal{C}_K^n$*.*

Such a frame $\phi_K(n)$ is called a n-saturation of K*, and its image,* i.e. $img(\phi_K(n))$*, is uniquely defined.*

Intuitively, the attacker knowledge no longer grows once the frame $\phi_K(n)$ has been reached. Then two processes $P \mid K$ and $Q \mid K$ are in trace equivalence for some initial knowledge $\phi_{\mathsf{hd}}(n_0)$ if, and only if, P' and Q' are in trace equivalence with an initial knowledge enriched with $\phi_K(n_0)$ and P' and Q' are the instantiations of P and Q considering $2n_0$ agents (n_0 honest agents and n_0 dishonest ones).

Proposition 2. *Consider $2n$ processes of the form ($1 \leq i \leq n$):*

$$P_i' = !\,\mathsf{new}\, c_i'.\mathsf{out}(c_i, c_i').\mathsf{in}(c_i', \mathsf{x}_i^1(z_i^1)).\ldots.\mathsf{in}(c_i', \mathsf{x}_i^{k_i}(z_i^{k_i})).1{:}\, P_i(z_i^1, \ldots, z_i^{k_i})$$

$$Q_i' = !\,\mathsf{new}\, c_i'.\mathsf{out}(c_i, c_i').\mathsf{in}(c_i', \mathsf{x}_i^1(z_i^1)).\ldots.\mathsf{in}(c_i', \mathsf{x}_i^{k_i}(z_i^{k_i})).1{:}\, Q_i(z_i^1, \ldots, z_i^{k_i})$$

where each P_i (resp. Q_i) is a basic process built on c_i', and $\mathsf{x}_i^j \in \{\mathsf{ag}, \mathsf{hon}, \mathsf{dis}\}$ for any $1 \leq j \leq k_i$, and the c_i for $1 \leq i \leq n$ are pairwise distinct. Moreover, we assume that ag, hon *and* dis *do not occur in P_i, Q_i ($1 \leq i \leq n$). Let $n_0 \in \mathbb{N}$, and K be a key generator such that $fc(K) \cap \{c_1, \ldots, c_n\} = \emptyset$. We have that:*

$$(K \uplus \{P'_i | 1 \leq i \leq n\};\ \phi_{\mathsf{hd}}(n_0);\ 0) \approx (K \uplus \{Q'_i | 1 \leq i \leq n\};\ \phi_{\mathsf{hd}}(n_0);\ 0)$$
if, and only if,
$$(\textstyle\bigcup_{i=1}^{n} \mathcal{P}_i;\ \phi_{\mathsf{a}}(n_0) \uplus \phi_K(n_0);\ 0) \approx (\bigcup_{i=1}^{n} \mathcal{Q}_i;\ \phi_{\mathsf{a}}(n_0) \uplus \phi_K(n_0);\ 0)$$

where $\phi_K(n_0)$ is a n-saturation of K, and

$\mathcal{P}_i = \{!\,\mathsf{new}\, c'_i.\mathsf{out}(c^i_{z_i^1,\ldots,z_i^{k_i}}, c'_i).1 : P_i(z_i^1, ..., z_i^{k_i}) | \mathsf{x}_i^1(z_i^1), ..., \mathsf{x}_i^{k_i}(z_i^{k_i}) \in img(\phi_{\mathsf{hd}}(n_0))\}$;

$\mathcal{Q}_i = \{!\,\mathsf{new}\, c'_i.\mathsf{out}(c^i_{z_i^1,\ldots,z_i^{k_i}}, c'_i).1 : Q_i(z_i^1, ..., z_i^{k_i}) | \mathsf{x}_i^1(z_i^1), ..., \mathsf{x}_i^{k_i}(z_i^{k_i}) \in img(\phi_{\mathsf{hd}}(n_0))\}$.

Example 13. Using more conventional notations for agent names and after applying Proposition 2, we deduce the following equivalence:

$$(\mathcal{P}_{\mathsf{DS}} \uplus \mathcal{P}'_1;\ \phi_0;\ 0) \approx (\mathcal{P}_{\mathsf{DS}} \uplus \mathcal{P}'_2;\ \phi_0;\ 0)$$

where

- $\phi_0 = \{\mathsf{w}_a \triangleright a;\ \mathsf{w}_b \triangleright b;\ \mathsf{w}_c \triangleright c;\ \mathsf{w}_d \triangleright d;\ \mathsf{w}_{kc} \triangleright \mathsf{shk}_{\mathsf{s}}(c);\ \mathsf{w}_{kd} \triangleright \mathsf{shk}_{\mathsf{s}}(d)\}$;
- $\mathcal{P}_{\mathsf{DS}} = \left\{ \begin{array}{l} !\,\mathsf{new}\, c_1.\mathsf{out}(c_{A,z_A,z_B}, c_1).P_A(z_A, z_B) \\ |\,!\,\mathsf{new}\, c_2.\mathsf{out}(c_{B,z_A,z_B}, c_2).P_B(z_A, z_B) \\ |\,!\,\mathsf{new}\, c_3.\mathsf{out}(c_{S,z_A,z_B}, c_3).P_S(z_A, z_B) \end{array} \middle|\ z_A, z_B \in \{a, b, c, d\} \right\}$
- $\mathcal{P}'_i = \{\,!\,\mathsf{new}\, c'_2.\mathsf{out}(c'_{B,z_A,z_B}, c'_2).P'_i(z_A, z_B) \mid z_A, z_B \in \{a, b\}\,\}$.

This corresponds to the standard scenario with 2 honest agents and 2 dishonest ones when assuming that agents may talk to themselves.

3.4 Sketch of proof of Theorem 1

First, thanks to the fact that we consider action-deterministic processes, we can restrict our attention to the study of the following notion of trace inclusion, and this is formally justified by the lemma stated below.

Definition 10. *Let $\mathcal{C}$ and $\mathcal{C}'$ be two configurations. We say that $\mathcal{C}$ is* trace included *in $\mathcal{C}'$, written $\mathcal{C} \sqsubseteq \mathcal{C}'$, if for every $(\mathsf{tr}, \phi) \in \mathsf{trace}(\mathcal{C})$, there exists $(\mathsf{tr}', \phi') \in \mathsf{trace}(\mathcal{C}')$ such that $\mathsf{tr} = \mathsf{tr}'$, and $\phi \sqsubseteq_s \phi'$.*

Lemma 3. *Let $\mathcal{C}$ and $\mathcal{C}'$ be two action-deterministic configurations. We have $\mathcal{C} \approx \mathcal{C}'$, if, and only if, $\mathcal{C} \sqsubseteq \mathcal{C}'$ and $\mathcal{C}' \sqsubseteq \mathcal{C}$.*

Given two action-deterministic configurations $\mathcal{C}$ and $\mathcal{C}'$ such that $\mathcal{C} \not\sqsubseteq \mathcal{C}'$, a *witness* of non-inclusion is a trace tr for which there exists ϕ such that $(\mathsf{tr}, \phi) \in \mathsf{trace}(\mathcal{C})$ and:

- either there does not exist ϕ' such that $(\mathsf{tr}, \phi') \in \mathsf{trace}(\mathcal{C}')$ (intuitively, the trace tr cannot be executed in $\mathcal{C}'$);
- or such a ϕ' exists and $\phi \not\sqsubseteq_s \phi'$ (intuitively, the attacker can observe that a test succeeds in ϕ and fails in ϕ').

Second, we show that we can restrict our attention to witnesses of non-inclusion that have a special shape: in case a constant from Σ_{error} is emitted, this happens only at the very last step. In other words, this means that we

may assume that the rule Let-Fail is applied at most once, at the end of the execution. More formally, a term t is Σ_{error}*-free* if t does not contain any occurrence of error for any $\mathsf{error} \in \Sigma_{\mathsf{error}}$. This notion is extended as expected to frames, and traces.

Lemma 4. *Let P and Q be two action-deterministic protocols, and ϕ_0 and ψ_0 be two frames that are Σ_{error}-free. If $(P;\, \phi_0;\, 0) \not\sqsubseteq (Q;\, \psi_0;\, 0)$ then there exists a witness tr of this non-inclusion such that:*

- *either tr is Σ_{error}-free;*
- *or tr is of the form $\mathsf{tr}'.\mathsf{out}(c, \mathsf{error})$ with tr' Σ_{error}-free and $\mathsf{error} \in \Sigma_{\mathsf{error}}$.*

This lemma relies on the fact that else branches are simple: at best they yield the emission of a constant in Σ_{error} but they may not trigger any interesting process.

We can then prove our key result: it is possible to bound the number of agents needed for an attack. To formally state this proposition, we rely on the frame $\phi_{\mathsf{hd}(n)}$ as introduced in Definition 5. Theorem 1 then easily follows from Proposition 3.

Proposition 3. *Let $\mathcal{E}$ be a constructor theory, and P and Q be two action-deterministic protocols such that $(P;\, \phi_{\mathsf{hd}}(n);\, 0) \not\sqsubseteq (Q;\, \phi_{\mathsf{hd}}(n);\, 0)$ for some $n \in \mathbb{N}$. We have that*

$$(P;\, \phi_{\mathsf{hd}}(n)\rho;\, 0) \not\sqsubseteq (Q;\, \phi_{\mathsf{hd}}(n)\rho;\, 0)$$

for some $\mathcal{A}$-renaming ρ such that $\phi_{\mathsf{h}}(n)\rho$ (resp. $\phi_{\mathsf{d}}(n)\rho$) contains at most $2b(\mathcal{E})+1$ distinct agent names, and $\phi_{\mathsf{h}}(n)\rho$ and $\phi_{\mathsf{d}}(n)\rho$ do not share any name.

Proof. (sketch) Of course, when $n \leq 2b(\mathcal{E}) + 1$, the result is obvious. Otherwise, let tr be a witness of non-inclusion for $(P;\, \phi_{\mathsf{hd}}(n);\, 0) \not\sqsubseteq (Q;\, \phi_{\mathsf{hd}}(n);\, 0)$. Thanks to Lemma 4, we can assume that tr is either Σ_{error}-free or of the form $\mathsf{tr}'.\mathsf{out}(c, \mathsf{error})$ for some $\mathsf{error} \in \Sigma_{\mathsf{error}}$. This means that the trace tr can be executed from $(P;\, \phi_{\mathsf{hd}}(n);\, 0)$ without using the rule Let-Fail at least up to its last visible action.

Considering a renaming ρ_0 that maps any honest agent name h to h_0, and any dishonest agent name d to d_0, we still have that the trace $\mathsf{tr}\rho_0$ can be executed from $(P;\, \phi_{\mathsf{hd}}(n)\rho_0;\, 0)$ at least up to its last visible action. Indeed, this renaming preserves equality tests and the property of being a message. Now, to ensure that the trace can still not be executed in the Q side (or maintaining the fact that the test under consideration still fails), we may need to maintain some disequalities, and actually at most $b(\mathcal{E})$ agent names have to be kept unchanged for this (remember that our theory is $b(\mathcal{E})$-blockable). Moreover, in case P executes its else branch, we have also to maintain some disequalities from the P side, and again we need at most to preserve $b(\mathcal{E})$ agent names for that. We do not know whether those names for which we have to maintain distinctness correspond to honest or dishonest agents, but in any case considering $2b(\mathcal{E}) + 1$ of each sort is sufficient. □

The following example illustrates why we may need $2b(\mathcal{E}) + 1$ agents of a particular sort (honest or dishonest) to carry out the proof as explained above.

Example 14. We also consider two constants $\mathsf{error}_1, \mathsf{error}_2 \in \Sigma_{\mathsf{error}}$. In processes P and Q below, we omit the channel name for simplicity. We may assume that all input/outputs occur on a public channel c.

$$P = \mathsf{in}(\mathsf{hon}(x_1)).\mathsf{in}(\mathsf{hon}(x_2)).\mathsf{in}(\mathsf{hon}(x_3)).\mathsf{in}(\mathsf{hon}(x_4)).\mathsf{let}\, z_1 = \mathsf{eq}(x_1, x_2)\, \mathsf{in}\\ \mathsf{let}\, z_2 = \mathsf{eq}(x_3, x_4)\, \mathsf{in}\, 0\, \mathsf{else}\, \mathsf{out}(\mathsf{error}_1)\\ \mathsf{else}\, \mathsf{out}(\mathsf{error}_2)$$

The process Q is as P after having swapped the two tests, and the two constants error_1 and error_2.

$$Q = \mathsf{in}(\mathsf{hon}(x_1)).\mathsf{in}(\mathsf{hon}(x_2)).\mathsf{in}(\mathsf{hon}(x_3)).\mathsf{in}(\mathsf{hon}(x_4)).\mathsf{let}\, z_1 = \mathsf{eq}(x_3, x_4)\, \mathsf{in}\\ \mathsf{let}\, z_2 = \mathsf{eq}(x_1, x_2)\, \mathsf{in}\, 0\, \mathsf{else}\, \mathsf{out}(\mathsf{error}_2)\\ \mathsf{else}\, \mathsf{out}(\mathsf{error}_1)$$

We have that $P \not\approx Q$. To see this, we may consider a trace where x_1, x_2, x_3, and x_4 are instantiated using distinct agent names. However, any trace where $x_1 = x_2$ or $x_3 = x_4$ (or both), does not allow one to distinguish these two processes. It is thus important to block at least one agent name among x_1, x_2, and one among x_3, x_4. This will ensure that both P and Q trigger their first else branch. Then, the remaining agent names can be mapped to the same honest agent name. Thus, applying our proof technique we need $b + b + 1$ honest agent names (and here $b = 1$). Note however that a tighter bound may be found for this example since 2 distinct honest agent names are actually sufficient. Indeed, choosing $x_1 = x_3$ and $x_2 = x_4$ allows one to establish non-equivalence. But such a choice would not be found following our technique.

Actually, we can show that there is no attack that requires simultaneously $2b+1$ honest agents and $2b + 1$ dishonest agents. We could elaborate a tighter bound, at the cost of having to check more equivalences.

4 Tightness of our hypothesis

Our class of protocols is somewhat limited in the sense that we consider processes that are action-deterministic, with simple else branches, and constructor theories. The counter-examples developed in this section actually suggest that our hypotheses are tight. We provide impossibility results for protocols in case any of our hypotheses is removed, that is, we provide counter-examples for processes with complex else branches, or non constructor theories, or non action-deterministic protocols.

4.1 Complex else branches

A natural extension is to consider processes with more expressive else branches. However, as soon as messages emitted in else branches may rely (directly or indirectly) on some agent names, this may impose some disequalities between arbitrary many agent names. This negative result already holds for the standard secrecy property expressed as a reachability requirement.

Formally, we show that we can associate, to any instance of PCP (Post Correspondance Problem), a process P (that uses only standard primitives) such that P reveals a secret s for n agents if, and only if, the corresponding PCP instance has a solution of length smaller than n. Therefore computing a bound for the number of agents needed to mount an attack is as difficult as computing a bound (regarding the length of its smallest solution) for the PCP instance under study. Computing such a bound is undecidable since otherwise we would get a decision procedure for the PCP problem by simply enumerating all the possible solutions until reaching the bound.

Property 1. There is an execution $(P;\, \phi_{\mathsf{hd}}(n);\, 0) \xrightarrow{\mathsf{tr.out}(c,\mathsf{w})} (\mathcal{P};\, \phi \uplus \{\mathsf{w} \triangleright \mathsf{s}\};\, 0)$ if, and only if, the instance of PCP under study admits a solution of length at most n.

An instance of PCP over the alphabet A is given by two sets of tiles $U = \{u_i \mid 1 \leq i \leq n\}$ and $V = \{v_i \mid 1 \leq i \leq n\}$ where $u_i, v_i \in \mathsf{A}^*$. A solution of PCP is a non-empty sequence $i_1, \ldots, i_p$ over $\{1, \ldots, n\}$ such that $u_{i_1} \ldots u_{i_p} = v_{i_1} \ldots v_{i_p}$. Deciding whether an instance of PCP admits a solution is well-known to be undecidable, and thus there are instances for which a bound on the size of a solution is not computable. We describe here informally how to build our process P made of several parts. For the sake of clarity, we simply provide the informal rules of the protocol. It is then easy (but less readable) to write the corresponding process. First, following the construction proposed *e.g.* in [15], we write a process P_{PCP} that builds and outputs all the terms of the form:

$$\mathsf{enc}(\langle\langle u, v\rangle, \ell\rangle, k)$$

where $u = u_{i_1} \ldots u_{i_p}$, $v = v_{i_1} \ldots v_{i_p}$, and ℓ is a list of agent names of length p that can be encoded using pairs. The key k is supposed to be unknown from the attacker. This can be easily done by considering rules of the form (where concatenation can be encoded using nested pairs):

$$\mathsf{ag}(z),\ \mathsf{enc}(\langle\langle x, y\rangle, z_\ell\rangle, k) \to \mathsf{enc}(\langle\langle x.u_i, y.v_i\rangle, \langle z, z_\ell\rangle\rangle, k)$$

for any pair of tiles (u_i, v_i).
We then need to check whether a pair $\langle u, v\rangle$ embedded in the term $\mathsf{enc}(\langle\langle u, v\rangle, \ell\rangle, k)$ is a solution of PCP.

$$\mathsf{enc}(\langle\langle x, x\rangle, z\rangle, k),\ \mathsf{enc}(z, k_{\mathsf{diff}}) \ \to\ \mathsf{s}$$

Second, to build our counter-example, we write a process that relies on some else branches to ensure that a list ℓ is made of distinct elements. The idea is that $\mathsf{enc}(\ell, k_{\mathsf{diff}})$ is emitted if, and only if, elements in ℓ are distinct agent names.

$$\mathsf{ag}(x) \ \to\ \mathsf{enc}(\langle x, \bot\rangle, k_{\mathsf{diff}})$$

$$\mathsf{ag}(x),\ \mathsf{ag}(y),\ \mathsf{enc}(\langle x, z\rangle, k_{\mathsf{diff}}), \mathsf{enc}(\langle y, z\rangle, k_{\mathsf{diff}}) \xrightarrow{x \neq y} \mathsf{enc}(\langle x, \langle y, z\rangle\rangle, k_{\mathsf{diff}})$$

The first rule allows us to generate list of length 1 whereas the second rule gives us the possibility to build list of greater length, like $[a_1, a_2, \ldots, a_n]$ as soon as the sublists $[a_1, a_3, \ldots, a_n]$ and $[a_2, a_3, \ldots, a_n]$ have been checked, and a_1 and a_2 are distinct agent names. The rule $u \xrightarrow{t_1 \neq t_2} v$ is the informal description for the

following process: on input u and if $t_1 \neq t_2$ then emit v. This can be encoded in our framework as explained in Section 2.3.

The formalisation of these rules yields a process P that satisfies Property 1, and it is not difficult to write a proces P that satisfies in addition our action-determinisim condition. This encoding can be adapted to show a similar result regarding trace equivalence. We may also note that this encoding works if we consider an execution model in which agents are not authorised to talk to themselves. In such a case, we even do not need to rely explicitly on else branches.

4.2 Pure equational theories

We now show that it is actually impossible to bound the number of agents for non constructor theories. This impossibility result already holds for the standard equational theory $\mathsf{E}_{\mathsf{enc}}$: $\mathsf{dec}(\mathsf{enc}(x, y), y) = x$.

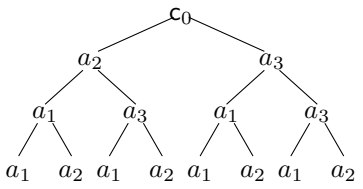
To prove our result, given a list ℓ of pairs of agent names, we build two terms $t^P(\ell)$ and $t^Q(\ell)$ using the function symbols enc, dec, the public constant c_0, and some agent names $a_1, \ldots, a_n$ in $\mathcal{A}$. The terms $t^P(\ell)$ and $t^Q(\ell)$ are such that they are equal as soon as two agent names of a pair in ℓ are identical.

Property 2. The terms $t^P(\ell)$ and $t^Q(\ell)$ are equal modulo $\mathsf{E}_{\mathsf{enc}}$ if, and only if, there exists a pair (a, b) in ℓ such that $a = b$.

The terms $t^P(\ell)$ and $t^Q(\ell)$ are defined inductively as follows:

- $t^P(\ell) = \mathsf{dec}(\mathsf{enc}(\mathsf{c}_0, a), b)$ and $t^Q(\ell) = \mathsf{dec}(\mathsf{enc}(\mathsf{c}_0, b), a)$ when $\ell = [(a, b)]$;
- In case $\ell = (a, b) :: \ell'$ with ℓ' non-empty, we have that
$$t^X(\ell) = \mathsf{dec}(\mathsf{enc}(\mathsf{c}_0, \mathsf{dec}(\mathsf{enc}(a, t_1), t_2)), \mathsf{dec}(\mathsf{enc}(b, t_1), t_2))$$
where m, t_1 and t_2 are such that $t^X(\ell') = \mathsf{dec}(\mathsf{enc}(\mathsf{c}_0, t_1), t_2)$ and $X \in \{P, Q\}$.

For illustration purposes, the term $t^P(\ell_0)$ for $\ell_0 = [(a_2, a_3), (a_1, a_3), (a_1, a_2)]$ is depicted below. A subtree whose root is labelled with n having subtrees t_1 and t_2 as children represents the term $\mathsf{dec}(\mathsf{enc}(n, t_1), t_2)$. The term $t^Q(\ell_0)$ is the same as $t^P(\ell_0)$ after permutation of the labels on the leaves. First, we may note that $t^P(\ell_0) = t^Q(\ell_0)$ when $a_1 = a_2$. Now, in case $a_1 = a_3$, we obtain $t^P(\ell_0) = t^Q(\ell_0) = \mathsf{dec}(\mathsf{enc}(\mathsf{c}_0, a_2), a_3)$, and we have that $t^P(\ell_0) = t^Q(\ell_0) = \mathsf{c}_0$ when $a_2 = a_3$. These are the only cases where $t^P(\ell_0)$ and $t^Q(\ell_0)$ are equal modulo $\mathsf{E}_{\mathsf{enc}}$. More generally, we can show that $t^P(\ell)$ and $t^Q(\ell)$ enjoy Property 2.



Now we may rely on these terms to build two processes P_n and Q_n such that $(P_n;\, \phi_{\mathsf{hd}}(n_0);\, 0) \not\approx (Q_n;\, \phi_{\mathsf{hd}}(n_0);\, 0)$ if, and only if, $n_0 \geq n$. These processes are as follows:
$$P_n = \mathsf{in}(c, \mathsf{ag}(z_1)) \ldots \mathsf{in}(c, \mathsf{ag}(z_n)).\mathsf{out}(c, t^P(\ell))$$
$$Q_n = \mathsf{in}(c, \mathsf{ag}(z_1)) \ldots \mathsf{in}(c, \mathsf{ag}(z_n)).\mathsf{out}(c, t^Q(\ell))$$

where ℓ is a list of length $n(n-1)/2$ which contains all the pairs of the form (z_i, z_j) with $i < j$.

Note that in case $n_0 < n$, in any execution, we are thus forced to use twice the same agent names, and thus the resulting instances of $t^P(\ell)$ and $t^Q(\ell)$ will be equal modulo $\mathsf{E}_{\mathsf{enc}}$. In case we have sufficiently many distinct agent names, the resulting instances of $t^P(\ell)$ and $t^Q(\ell)$ will correspond to distinct public terms. Hence, in such a case trace equivalence does not hold.

Note that, for sake of simplicity, our encoding directly relies on the agent names, but a similar encoding can be done using for instance $\mathsf{shk}_{\mathsf{s}}(a)$ instead of a so that agent names will not be used in key position.

4.3 Beyond action-deterministic processes

Another natural extension is to get rid of the action-determinism condition, or at least to weaken it in order to consider processes that are determinate (as defined *e.g.* in [10]). This is actually not possible. The encoding is quite similar to the one presented in Section 4.1. Since we have no easy way to ensure that all the terms of the form $\mathsf{enc}(\ell, k_{\mathsf{diff}})$ will contain distinct elements, the encoding is more involved.

To prove our result, we show that given an instance of PCP, it is possible to build two processes P and Q (that use only standard primitives and no else branch) that are in equivalence for n agents if, and only if, the corresponding PCP instance has a solution of length at most n.

Property 3. $(P;\, \phi_{\mathsf{hd}}(n);\, 0) \not\approx (Q;\, \phi_{\mathsf{hd}}(n);\, 0)$ if, and only if, the instance of PCP under study admits a solution of length at most n.

Our process P is quite similar to the one described in Section 4.1. Note that the test $x \neq y$ has been removed, and a public constant yes has been added inside each encryption. The presence of such a constant is not mandatory when defining P but will become useful when defining Q.

$$\mathsf{enc}(\langle\langle x, x\rangle, z\rangle, k),\ \mathsf{enc}(\langle z_b, z\rangle, k_{\mathsf{check}}) \xrightarrow{z_b = \mathsf{yes}} \mathsf{ok} \tag{1}$$

$$\mathsf{ag}(x) \longrightarrow \mathsf{enc}(\langle \mathsf{yes}, \langle x, \bot\rangle\rangle, k_{\mathsf{check}}) \tag{2}$$

$$\mathsf{ag}(x),\ \mathsf{ag}(y),\ \begin{array}{l}\mathsf{enc}(\langle z_b, \langle x, z\rangle\rangle, k_{\mathsf{check}}),\\ \mathsf{enc}(\langle z'_b, \langle y, z\rangle\rangle, k_{\mathsf{check}})\end{array} \longrightarrow \mathsf{enc}(\langle \mathsf{yes}, \langle x, \langle y, z\rangle\rangle\rangle, k_{\mathsf{check}}) \tag{3}$$

Then, Q is quite similar except that we replace the test $z_b = \mathsf{yes}$ by $z_b = \mathsf{no}$ and we consider in addition three other versions of the last protocol rule (rule (3)) giving us a way to generate encryption containing the flag no. More precisely, we consider the following rule with φ equal to $x = y$ (rule 3a), $z_b = \mathsf{no}$ (rule 3b), and $z'_b = \mathsf{no}$ (rule 3c).

$$\mathsf{ag}(x),\ \mathsf{ag}(y),\ \begin{array}{l}\mathsf{enc}(\langle z_b, \langle x, z\rangle\rangle, k_{\mathsf{check}}),\\ \mathsf{enc}(\langle z'_b, \langle y, z\rangle\rangle, k_{\mathsf{check}})\end{array} \xrightarrow{\varphi} \mathsf{enc}(\langle \mathsf{no}, \langle x, \langle y, z\rangle\rangle\rangle, k_{\mathsf{check}})$$

Putting all these rules together and considering randomised encryption to avoid spurious equalities to happen, this yields two processes P and Q that actually satisfy Property 3.

Proof sketch. ($\Leftarrow$) if PCP has a solution of length at most n, it is possible to build the term $\mathsf{enc}(\langle u, v\rangle, \ell\rangle, k)$ corresponding to this solution with $u = v$ and ℓ of length at most n. Moreover, we can assume that ℓ is made of distinct elements. Hence, the additional rules in Q will not be really useful to generate a certificate on the list ℓ with the flag set to no. Actually, only $\mathsf{enc}(\langle \mathsf{yes}, \ell\rangle, k_{\mathsf{check}})$ will be generated, and thus P will emit ok and Q will not be able to mimic this step.

($\Rightarrow$) Now, if PCP has no solution of length at most n, then either PCP has no solution at all, and in such a case, the part where P and Q differ is not reachable, and thus the processes are in trace equivalence. Now, assuming that PCP has a solution of length n' with $n' > n$, the only possibility to distinguish P from Q is to build the term $\mathsf{enc}(\langle \mathsf{yes}, \ell\rangle, k_{\mathsf{check}})$ with ℓ of length n'. This term will allow us to trigger the rule (1) in P but not in Q. The problem is that ℓ contains a duplicate entry, and due to this, at some point it would be possible to mimic what is done in P using rule (3) with the additional rule ($3a$), and to pursue the construction of this certificate relying on ($3b$) and ($3c$). This will allow Q to go through the rule (1) as P did. □

5 Conclusion

We have shown that we can bound the number of agents for a large class of protocols: action-deterministic processes with simple else branches and constructor theories, which encompasses many primitives. The resulting bound is rather small in general. For example, 4 agents are sufficient for standard primitives and processes without else branches. Our assumptions are rather tight. Surprisingly, such a reduction result does not hold in case processes are not action-deterministic, or if they include more complex else branches, or else for more general equational theories. This draws a thin line between our result (where terms with destructors may not be sent) and a more general framework.

Our result applies for any equivalence between two processes. This allows us to cover various security properties such as strong secrecy or anonymity. However, assuming deterministic processes discards the encoding of some properties such as unlinkability. We devise in [17] an alternative encoding to check for unlinkability in our framework, considering only deterministic processes.

Our reduction result enlarges the scope of some existing decidability results. For example, [13] provides a decision procedure for an unbounded number of sessions, for processes that use at most one variable per rule. In case an arbitrary number of agents is considered, one or two variables are typically used simply to describe the agents. Bounding the number of agents is therefore needed to consider non trivial protocols.

The proof of our reduction result is inspired from [14], which shows how to bound the number of nonces. Taking advantage of the properties of agent names, we extend [14] to processes with simple else branches, action-determinism and general constructor theories. As future work, we plan to study how to generalize both results in a framework that would allow to bound several types of data.

References

1. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proceedings of the 28th Symposium on Principles of Programming Languages (POPL'01)*. ACM Press, 2001.
2. M. Arapinis, T. Chothia, E. Ritter, and M. Ryan. Analysing unlinkability and anonymity using the applied pi calculus. In *Proceedings of the 23rd Computer Security Foundations Symposium (CSF'10)*, pages 107–121. IEEE Computer Society Press, 2010.
3. A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA Tool for the automated validation of internet security protocols and applications. In K. Etessami and S. Rajamani, editors, *Proceedings of the 17th International Conference on Computer Aided Verification, CAV'2005*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285, Edinburgh, Scotland, 2005. Springer.
4. M. Backes, M. Maffei, and D. Unruh. Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol. In *Proceedings of 29th IEEE Symposium on Security and Privacy*, May 2008.
5. D. Baelde, S. Delaune, and L. Hirschi. Partial order reduction for security protocols. In L. Aceto and D. de Frutos-Escrig, editors, *Proceedings of the 26th International Conference on Concurrency Theory (CONCUR'15)*, volume 42 of *Leibniz International Proceedings in Informatics*, pages 497–510, Madrid, Spain, Sept. 2015. Leibniz-Zentrum für Informatik.
6. B. Blanchet. Proverif 1.91. http://prosecco.gforge.inria.fr/personal/bblanche/. As downloaded on October 1st, 2015. See files in directory /examples/pitype/choice/.
7. B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proc. of the 14th Computer Security Foundations Workshop (CSFW'01)*. IEEE Computer Society Press, June 2001.
8. B. Blanchet. An automatic security protocol verifier based on resolution theorem proving (invited tutorial). In *Proceedings of the 20th International Conference on Automated Deduction (CADE-20)*, Tallinn, Estonia, July 2005.
9. B. Blanchet, M. Abadi, and C. Fournet. Automated verification of selected equivalences for security protocols. *Journal of Logic and Algebraic Programming*, 75(1):3–51, Feb.–Mar. 2008.
10. R. Chadha, Ş. Ciobâcă, and S. Kremer. Automated verification of equivalence properties of cryptographic protocols. In *Programming Languages and Systems — Proceedings of the 21th European Symposium on Programming (ESOP'12)*, volume 7211 of *Lecture Notes in Computer Science*, pages 108–127. Springer, Mar. 2012.
11. V. Cheval, V. Cortier, and S. Delaune. Deciding equivalence-based properties using constraint solving. *Theoretical Computer Science*, 492:1–39, 2013.
12. T. Chothia and V. Smirnov. A traceability attack against e-passports. In *Proceedings of the 14th International Conference on Financial Cryptography and Data Security (FC 2010)*, 2010.
13. R. Chrétien, V. Cortier, and S. Delaune. From security protocols to pushdown automata. In *Proceedings of the 40th International Colloquium on Automata, Languages and Programming (ICALP'13)*, volume 7966 of *Lecture Notes in Computer Science*, pages 137–149. Springer, July 2013.

14. R. Chrétien, V. Cortier, and S. Delaune. Checking trace equivalence: How to get rid of nonces? In *Proceedings of the 20th European Symposium on Research in Computer Security (ESORICS'15)*, Lecture Notes in Computer Science, Vienna, Austria, 2015. Springer.
15. H. Comon-Lundh and V. Cortier. Security properties: Two agents are sufficient. *Science of Computer Programming*, 50(1-3):51–71, Mar. 2004.
16. H. Comon-Lundh and V. Cortier. Computational soundness of observational equivalence. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS'08)*, pages 109–118, Alexandria, Virginia, USA, Oct. 2008. ACM Press.
17. V. Cortier, A. Dallon, and S. Delaune. Bounding the number of agents, for equivalence too. Research Report LSV-16-01, Laboratoire Spécification et Vérification, ENS Cachan, France, Jan. 2016.
18. C. Cremers. The Scyther Tool: Verification, falsification, and analysis of security protocols. In *Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, USA, Proc.*, volume 5123/2008 of *Lecture Notes in Computer Science*, pages 414–418. Springer, 2008.
19. S. Delaune, S. Kremer, and M. D. Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, (4):435–487, July 2008.
20. D. Denning and G. Sacco. Timestamps in key distributed protocols. *Communication of the ACM*, 24(8):533–535, 1981.
21. B. Schmidt, S. Meier, C. Cremers, and D. Basin. Automated analysis of Diffie-Hellman protocols and advanced security properties. In S. Chong, editor, *Proceedings of the 25th IEEE Computer Security Foundations Symposium, CSF 2012, Cambridge, MA, USA, June 25-27, 2012*, pages 78–94. IEEE, 2012.