Verification of cryptographic protocols : A bound on the number of agents

Antoine Dallon

02/10/2015

Supervisors :
Stéphanie Delaune (Team SECSI - LSV)
Véronique Cortier (Team Cassis - LORIA)

General context

Nowadays, cryptography and security protocols are used everywhere, and public confidence in credit cards or electronic voting relies on their trustworthiness. As flaws can stay unremarked during several years, the scientific community has undertaken the automation of their research through formal methods, or to prove the protocols secure. Two families of models have been introduced : computational models are based on real properties of cryptographic primitives whereas symbolic models, which I used, assume perfect cryptography but allow more automation.

In each model, security properties should hold for an arbitrary number of agents involved in an unbounded number of sessions. This together with the unbounded size of the messages, makes the verification problem undecidable. Works verification of security properties in the symbolic model search for decidable classes of protocols and properties, or try to improve automated research of flaws.

Research problem

The goal of my internship was to bound the number of agents implied in an attack. More precisely, it was to show that there was a small computable number such that for any protocol of some class, if there exists an attack, then there exists an attack involving less than this number of agents. Such a bound allows us to forget about universal quantification about the agents, reducing the set of possible traces of attacks. It helps to focus on other difficulties. This problem had been solved ten years ago for some properties, like secrecy and authentication, and my work aimed to extend this result to more recent properties expressed using a notion of equivalence, like strong secrecy and different flavours of anonymity. Actually, existing work on verification of equivalence-based security protocols does not take account of an unbounded number of agents. So it seemed natural to ask if and when it was safe to do so.

Contribution

To answer this question, I studied related results. One of those stated that nonces can soundly be abstracted away for the class of simple protocols (see [6]). I showed that it was possible to deduce a bound on the number of agents for this class of protocols. Then my goal was to extend this class. This research led me to several negative results : some extensions were impossible, as shown by some counter examples I built : in some cases, it was impossible to project agents on others. These examples gave me a better understanding of the previous proof, enabling me to find more suitable hypothesis. It was then possible to prove an extension where a new form of the unlinkability property can be stated. To do that, I had to give a new unlinkability property. I proved with ProVerif that this hypothesis was consistent with the known flaw on French passport.

Arguments

The validity of the result is based on the proofs I wrote. The main difficulty was to find the agents that we need to preserve, whereas others may be projected on the same one. To do that, each hypothesis is necessary, not only because our proof does not work without them, but also because removing these hypothesis allows to build counter-examples. These counter examples are interesting in themselves, because they give impossibility results for several extensions.

Moreover, these hypothesis are sufficient to encompass most of real protocols. For these protocols, the bound is often the lowest possible, that is three for each kind of agents (honest or dishonest), which is a small number.

Summary and future work

My approach for the names of agents was inspired by a nonce abstraction soundness result for the class of simple protocols. The result I have shown is more general in the sense that it is true for a more general class of protocols, and more precise because it includes completeness. But my theorem holds for names of agents only. So it seems natural to unify these results through a more general theorem. It seems that the soundness part could also be true for other kinds of data, like public constants. As the completeness statement is false for nonces, a natural question is to understand when it is true. Moreover, some real protocols containing else branches are not in our class, but it is not sure that the result could not be true for those protocols. An important related open problem is to bound the number of sessions, but it is a lot more difficult and cannot be handled the same way.

1 Introduction

Security protocols are distributed programms designed to ensure security properties using cryptographic primitives. Some of these protocols have been found to be flawed years after their deployment, even for simple kinds of attacks. This has led to the need to automatically verify protocols using formal methods. Symbolic models have been proposed, e.g. by Dolev and Yao, where cryptography is seen as black box. Some tools, e.g. ProVerif [4], have been developped to check properties in such models. However, as the problem of verification of security properties has been shown to be undecidable for an unbounded number of sessions, e.g. in [11] and [12], they either rely on non-complete approaches or bound arbitrarly the number of sessions.

A typical formal model is the applied pi-calculus [1]. This process algebra uses terms that are used to represent cryptographic operations (e.g. symmetric and asymmetric encryption and decryption, signature). The security properties are either expressed as trace properties, that is properties that should hold for any execution (such as secrecy or authentication), or as equivalence properties, when two protocols should be indistinguishable. This second kind of properties may be used to ask typically that some protocol is as sure as an idealized version, or that it is impossible to distinguish when a protocol is used by Alice or by Bob. These equivalence properties allow to state e.g. stronger notions of secrecy, anonymity in private authentication or in electronic voting.

When it is claimed that some protocol ensures some security property, this property should hold for an unbounded number of agents involved in an unbounded number of sessions. It has been shown that for trace properties and for some class of protocols, it is sufficient to study the protocol with a bounded number of agents (see [9]). For trace properties, the bound on the necessary number to verify secrecy is two (one honest and one dishonest agent) because what is done by an honest agent can be played by another. The goal of the internship was to bound the number of agents for equivalence properties. More precisely, it was to find a class of protocols such that between any protocols of this class, the equivalence holds with any number of agents iff it holds with a bounded number of agents, where the bound is computable from the protocol.

However, for equivalence properties, we cannot hope to get two agents as a bound, because disequivalence between protocols may come from a disequality. For example, the two terms $\mathrm{adec}(\mathrm{aenc}(m, \mathrm{pk}(ag_1)), \mathrm{sk}(ag_2))$ and m are distinguable iff $ag_1 \neq ag_2$, where adec and aenc represent asymmetric decryption and encryption respectively, ag_1 and ag_2 two names of agent, and $(\mathrm{pk}(x), \mathrm{sk}(x))$ the pair of the public key and the private key of agent x. My main contribution is to show that to prove equivalence between protocols of some class, six agents are sufficient.

Contribution. The first step was to model sessions between an arbitrary number of agents. For this, I introduce a special frame (sequence of messages) of arbitrary size, that instanciate participant's names. Then I show that (for some class of protocols), there exists an attack against equivalence with some number of agents iff there exists an attack with some bounded and calculable number of agents. The proof is inspired from [6], where it is shown that it is possible to abstract away nonces. This allows to forget about the universal quantification on agents, even for an unbounded number of sessions and an unbounded number of nonces. Most cryptographic primitives are handled, included symmetric and asymmetric encryption, signature, hash functions, MACs, provided that they can be expressed as reduction rule of a destructor on constructor terms. The bound on the number of agents depends on these primitives, the relevant parameter being the size of the maximal set of unifiable-together rules. For standard primitives, this parameter is one, giving a minimal number of agents of 6 (3 honest and 3 dishonest). But this number can be computed on any equational theory where rules are destructor on constructor terms.

The considered class of protocols is a class of *action-deterministic* protocols, that is protocols where two indistinguishable actions cannot be reachable at the same time. They have first been defined in [3], and they contain simple protocols, where each message is identified by a public session identifier. In our model, we allow simple else branches containing only an output of a public error. This is sufficient to model several practical examples such as the biometric passport. Moreover, "if" tests can be done on disjonctions of conjonctions of equalities. It is quite rare that results on equivalence properties hold for a class of protocols with else branches or disjunctive conditions. These tests are encoded in the theory : we have some eq_n destructors that test if some equality among n is true and reduce in this case. Using these destructor increases the bound on the number of agents.

Finally, I consider possible extensions of the result, and I provide several counter-examples. When general else branches are allowed, it is not possible to get such a result, because tests of difference can increase arbitrarly the bound. With pure equational theories, it is possible to build terms that are different iff the agents of a list of arbitrary size are pairwise distinct. When no notion of determinism is used, it is possible to build protocols where a frame can have several executions that are different iff an arbitrary number of agents is involved.

2 Model

Protocols are modelled as processes inspired from the applied pi-calculus [1]. They manipulate terms that represent messages.

2.1 Term algebra

We assume three infinite sets $\hat{\mathcal{N}}$, $\mathcal{X}$, $\mathcal{W}$. The set $\hat{\mathcal{N}}$ is the set of *names*. Its elements represent e.g. nonces, keys and agents. We denote $\hat{\mathcal{N}} = \mathcal{N} \uplus \mathcal{A}$ where $\mathcal{A}$ is the set of the names of agents, and is also split in two parts : $\mathcal{A} = \mathcal{A}^H \uplus \mathcal{A}^D$ the set of the names of dishonest and honest agents. We assume that $\mathcal{A}^H = \{ag_1^H, ag_2^H, \ldots\}$ and $\mathcal{A}^D = \{ag_1^D, ag_2^D, \ldots\}$. Sometimes, it will be useful to consider only a finite number of agent names. For any integer k, we denote $\mathcal{A}_k^H = \{ag_1^H, \ldots, \mathrm{ag}_k^H\}$, $\mathcal{A}_k^D = \{ag_1^D, \ldots, ag_k^D\}$ and $\mathcal{A}_k = \mathcal{A}_k^H \uplus \mathcal{A}_k^D$.

$\mathcal{X}$ is a set of variables, that refer to data that are learnt by the participants during the execution of the process or that allow the attacker to instanciate the processes (e.g. with identities of agents). The set $\mathcal{W}$ is a set of variables, disjoint from $\mathcal{X}$, that are used to represent attacker's knowledge.

Moreover, we assume a signature Σ, that is a set of function symbols together with their arity. The elements of Σ are split between the set of *constructor* symbols Σ_c and *destructor* symbols Σ_d : $\Sigma = \Sigma_c \uplus \Sigma_d$. The set of terms built on data from the set A (e.g. $A = \hat{\mathcal{N}} \cup \mathcal{X}$, $A = \hat{\mathcal{N}}$ or $A = \mathcal{W}$) from symbols in Σ is denoted by $\mathcal{T}(\Sigma, A)$.

The set $\mathcal{T}(\Sigma_c, \mathcal{X} \cup \hat{\mathcal{N}})$ of terms built on names and variables in $\mathcal{X}$ is the set of *constructor terms*. The terms of $\mathcal{T}(\Sigma, \hat{\mathcal{N}})$ are called *ground*. We consider some special subset $\mathcal{M}_\Sigma$ of ground constructor terms, that is $\mathcal{M}_\Sigma \subset \mathcal{T}(\Sigma_c, \hat{\mathcal{N}})$. Moreover, we assume that $\mathcal{M}_\Sigma$ and $\mathcal{T}(\Sigma_c, \hat{\mathcal{N}}) \backslash \mathcal{M}_\Sigma$ are stable by renaming of names and constants. $\mathcal{M}_\Sigma$ is called the set of messages.

Example 1 : We want to modelize symmetric cryptography as a black box. So we will consider two symbols, enc and dec, where enc represents encryption and dec represents decryption. The symbol $\mathrm{shk}/1$ will be used to represent a key shared with a server.

So we consider the signature $\Sigma = \{\mathrm{enc}/2; \mathrm{dec}/2; \mathrm{shk}/1; ok/0\}$ with constructors $\Sigma_c = \{\mathrm{enc}/2; \mathrm{shk}/1; ok/0\}$ and $\Sigma_d = \{\mathrm{dec}/2\}$. Assume that $K_{ab} \in \mathcal{N}$, $a, b \in \mathcal{A} \subset \hat{\mathcal{N}}$. We have that

$\mathrm{dec}(\mathrm{enc}(a, \mathrm{shk}(b)), \mathrm{shk}(b)) \in \mathcal{T}(\Sigma, \hat{\mathcal{N}})$, but it is not a constructor term. The term $\mathrm{enc}(<< b, K_{ab} >, \mathrm{enc}(< K_{ab}, a >, \mathrm{shk}(b)) >, \mathrm{shk}(a))$ is a constructor term.

We make also a distinction between *public* and *private* symbols : $\Sigma = \Sigma_{pub} \uplus \Sigma_{priv}$. The public symbols are those that are available to the attacker. The private symbols are symbols that are not public and thus not available to the attacker. A term can be built by the attacker by a recipe $R \in \mathcal{T}(\Sigma_{pub}, \mathcal{W})$ where the elements of $\mathcal{W}$ refer to messages the attacker knows. As names are initially unknown from the attacker (recall that they refer to nonces and keys), recipes only refer to $\mathcal{W}$ and not to $\mathcal{N}$.

Example 2 : We want to modelize the fact that the attacker cannot know the keys of the agents. So, we modelize the key as a private symbol. Consider the signature $\Sigma = \{\mathrm{enc}/2; \mathrm{dec}/2; \mathrm{shk}/1; ok/0\}$ with public symbols $\Sigma_{pub} = \{\mathrm{enc}/2; \mathrm{dec}/2; ok/0\}$ and $\Sigma_{priv} = \{\mathrm{shk}/1\}$. Let $w_1, w_2 \in \mathcal{W}$. $R = \mathrm{dec}(w_1, w_2)$ is a recipe because dec is a public symbol, but $\mathrm{shk}(w_1)$ is not because shk is private. An attacker may decrypt a message refered by w_1 with a key refered by w_2, but he cannot deduce the key of an agent refered by w_1.

Σ_0 is the set of elements of arity zero, that we call *constants*. We denote $\Sigma_0^{pub} = \Sigma_0 \cap \Sigma_{pub}$ the set of public constants. In this set, there is some special subset $\Sigma_0^{err} \subset \Sigma_0^{pub}$ of *errors*. If $\tilde{\Sigma}$ is some subset of Σ, we denote $\tilde{\Sigma}^- = \tilde{\Sigma} \backslash \Sigma_0^{err}$ the set of non-error symbols of $\tilde{\Sigma}$. For example, Σ_0^- is the set of non-error constants and Σ^- the set of non-error symbols. We assume that the set of non-error public constants, that is $(\Sigma_0^{pub})^-$, is infinite. It allows the attacker to use as much constants he wants.

Example 3 : Consider the signature $\Sigma = \{\mathrm{enc}/2; \mathrm{dec}/2; \mathrm{shk}/1; MacError/0\} \cup \{a_0/0; a_1/0; a_2/0; \ldots\}$ with errors $\Sigma_0^{err} = \{MacError/0\}$.
We have $\Sigma_0 = \{MacError/0; a_0/0; a_1/0; \ldots\}$ and $\Sigma_0^- = \{a_0/0; a_1/0; \ldots\}$.

When σ is a substitution, we denote by $dom(\sigma)$ its *domain*. Moreover, let u be a term. $u\sigma$ is the application of σ on u. We denote by $vars(u)$ the set of variables that occur in u. If u is ground, $vars(u) = \emptyset$ and $u\sigma = u$ for any substitution σ. We say that the terms u and v are *unifiable* when there is a substitution such that $u\sigma = v\sigma$.

Example 4 : Consider the signature :

$$\Sigma = \{\mathrm{enc}/2; \mathrm{dec}/2; \mathrm{shk}/1\} \cup (\Sigma_0^{pub})^-$$

We define $u = \mathrm{enc}(<< b, y_{ab} >, y_{bs} >, \mathrm{shk}(a))$ and a substitution :

$$\sigma = \{y_{ab} \triangleright K_{ab}; y_{bs} \triangleright \mathrm{enc}(< K_{ab}, a >, \mathrm{shk}(b))\}$$

We have $u\sigma = \mathrm{enc}(<< b, K_{ab} >, \mathrm{enc}(< K_{ab}, a >, \mathrm{shk}(b)) >, \mathrm{shk}(a))$.

The positions of a term t are defined as usual. Moreover, we also define $t|_p$ as the subterm of t at position p and $t[t']_p$ as the term t where the subterm at position p has been replaced by t'.

Example 5 : If $\Sigma = \{\mathrm{enc}/2; \mathrm{dec}/2; m/0\}$ and $u = \mathrm{enc}(<< b, y_{ab} >, y_{bs} >, \mathrm{shk}(a))$, then $u|_{1.1.2} = y_{ab}$ and $u[K_{ab}]_{1.1.2} = \mathrm{enc}(<< b, K_{ab} >, y_{bs} >, \mathrm{shk}(a))$.

To represent cryptographic operations, we use *rewriting rules*. A rewriting rule is a rule $g(t_1, \ldots, t_n) \to t$ where $g \in \Sigma_d$ and $t_1, \ldots, t_n, t \in \mathcal{T}(\Sigma_c^-, \mathcal{X})$. We say that u *rewrites into* v if there is a position $p \in pos(u)$, a rule $g(t_1, \ldots, t_n) \to t$ and a substitution θ such that $u|_p = g(t_1, \ldots, t_n)\theta$ and $v = u[t\theta]_p$ with $t_1\theta, \ldots, t_n\theta \in \mathcal{M}_\Sigma$. In particular, note that rewriting rules can only apply when subterms corresponding to $t_1, \ldots, t_n$ are constructor terms (eventually with errors).

Example 6 : We take $\Sigma_c = \{\text{enc}/2\} \cup (\Sigma_0^{pub})^-$, $\Sigma_d = \{\text{dec}/2\}$ together with the rewriting rule $\text{dec}(\text{enc}(x,y),y) \rightarrow x$. Let k_0 and $k_1 \in \mathcal{N}$. Then $\text{dec}(\text{enc}(m,k_0),k_0)$ rewrites into m but $\text{dec}(\text{enc}(m,\text{dec}(m,m)),\text{dec}(m,m))$ doesn't rewrite into m because $\text{dec}(m,m)$ does not rewrites into a constructor term.

We only consider sets of rewriting rules that yield convergent rewriting systems (see [10]), and we denote by $u\!\downarrow$ the normal form of a given term u.

2.2 Process algebra

Definition 1 (Process) : *Let Ch be an infinite set of channels. The processes we consider are built using the following grammar :*

$$P,Q = 0 \mid !^{c}_{c'}P \mid (P|Q) \mid \text{in}(c,u).P \mid \text{out}(c,u).P$$

$$\mid \text{new}\, n.P \mid \text{let}\, y = v \,\text{in}\, P \,\text{else}\, E \mid \text{phase}\, t.P$$

where t is an integer, u is a constructor term ($u \in \mathcal{T}(\Sigma_c, \hat{\mathcal{N}} \cup \mathcal{X})$), v is a term ($v \in \mathcal{T}(\Sigma, \hat{\mathcal{N}} \cup \mathcal{X})$), $n \in \mathcal{N}$ is a name, c, c' are channels of Ch and the process E is either 0 or $\text{out}(c,e)$*, with e an error ($e \in \Sigma_0^{err}$).*

We can define some subclass of restricted processes where errors only occur in else branches, and agent's name do not occur. They are built on the following grammar :

$$P,Q = 0 \mid !^{c}_{c'}P \mid (P|Q) \mid \text{in}(c,u).P \mid \text{out}(c,u).P$$

$$\mid \text{new}\, n.P \mid \text{let}\, y = v \,\text{in}\, P \,\text{else}\, E \mid \text{phase}\, t.P$$

where t is an integer, u is a non-error *constructor term* without agent's name *($u \in \mathcal{T}(\Sigma_c^-, \mathcal{N} \cup \mathcal{X})$), v is a non-error term* without agent's name *($v \in \mathcal{T}(\Sigma^-, \mathcal{N} \cup \mathcal{X})$), $n \in \mathcal{N}$ is a name, c, c' are channels of Ch and the process E is either 0 or* $\text{out}(c,e)$*, with e an error ($e \in \Sigma_0^{err}$).*

Note that the absence of the if construction is not a real limitation as $\text{if}\, y = y' \,\text{then}\, P \,\text{else}\, E$ can be encoded as $\text{let}\, x = \text{eq}(y,y') \,\text{in}\, P \,\text{else}\, E$ where $\text{eq}/2$ is a destructor, $\text{ok}/0$ is a constructor and $\text{eq}(x,x) \rightarrow \text{ok}$ is the only rule containing eq. So removing the if is without loss of generality.

Let describe informally the roles of the elements. The process 0 symbolizes the process doing nothing. Usually, we denote P instead of $P.0$. The process $!^{c}_{c'}P$ executes an arbitrary number of sessions of P (like $!P$ in pi-calculus) but the replication rule is a visible one and creates a new channel c' and declares it on the previous channel c. Such a notation has been introduced first in [3]. All channels are public, and we use channel generation to allow the attacker to know which session of the processes he is executing. The process $(P|Q)$ is used to represent the process executing P and Q in parallel. $\text{new}\, n.P$ renames n in P to model new elements unknown by the attacker (e.g. fresh keys, fresh nonces and new agents names). $\text{in}(c,u).P$ is a process expecting a message m unifiable with u on channel c and instantiating u by m in P. $\text{out}(c,u).P$ is a process outputing u on channel c and then executing P. The process $\text{let}\, x = v \,\text{in}\, P \,\text{else}\, E$ is a process trying to reduce v into a message. If it succeeds, then x is instantiated by $v\!\downarrow$ in P and P is executed. If it fails, E is executed. $\text{phase}\, t.P$ is a process that can only be executed later, when the phase becomes equal to t. When the phase advances, the processes in previous phase can't be executed anymore. The channels of Ch are either used in the specification or used to instantiate the channels during the execution. So we denote Ch_0 an infinite set of channels

that may be used in specifications of protocols and Ch^{fresh} an infinite set of channels that can only be used to instantiate channels during the execution. We have $Ch = Ch_0 \uplus Ch^{fresh}$.

We say that a variable $x \in \mathcal{X}$ is *free* if it is not in the scope of any "let" or "in" instruction. A process is *ground* if it has no free variable. A name $n \in \mathcal{N}$ is a *free name* if it is not in the scope of any new instruction. A channel $c \in Ch$ is free if it is not in the scope of any replication (that is, no $!_c^{c'}$ occurs for any c', but it is possible that some $!_{c'}^{c}$ occurs). Note that a ground process may have free names and free channels. We may note $P(x)$ a process where x is a free variable and $P(a) = P(x)\sigma$ where $\sigma = \{x \triangleright a\}$ and a is a constant.

Definition 2 : *A (restricted) protocol P is a ground (restricted) process that only uses channels of Ch_0.*

Example 7 : The Denning Sacco protocol [8] is a key distribution protocol. It can be described informally as follows :

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 S is a trusted server, K_{as} and K_{bs} are two keys shared by A and B respectively with the server : they will be modelized by $\mathrm{shk}(a)$ and $\mathrm{shk}(b)$. K_{ab} a key created by the server and $\{m\}_k$ represents the message m encrypted by the key k.

More formally, consider the signature :

$$\Sigma = \{\mathrm{enc}\,/2; \mathrm{dec}\,/2; <,>\,/2; \mathrm{shk}\,/1; \mathrm{proj}_1\,/1; \mathrm{proj}_2\,/1\} \cup (\Sigma_0^{pub})^-$$

with all symbols public, destructor symbols $\mathrm{dec}, \mathrm{proj}_i$ $(i = 1, 2)$, and rewriting rules $\mathrm{dec}(\mathrm{enc}(x, y), y) \to x$ and $\mathrm{proj}_i(< x_1, x_2 >) \to x_i$ $(i = 1, 2)$.

Consider two agents names a, b (for example $a = ag_1^H$ and $b = ag_2^H$) and define

$$P_A(a, b) = \mathrm{out}(c'_A, < a, b >).\, \mathrm{in}(c'_A, \mathrm{enc}(<< b, y_{ab} >, y_{bs} >, \mathrm{shk}(a)).\, \mathrm{out}(c'_A, y_{bs})$$

with $c'_A \in Ch_0$, $y_{ab}, y_{bs} \in \mathcal{X}$. P_A is a protocol, but it is not a restricted protocol because agent's names are used.

We can also define :

$$P_B(a, b) = \mathrm{in}(c'_B, \mathrm{enc}(< z_{ab}, a >, \mathrm{shk}(b)))$$

where $c_B \in Ch_0$, $z_{ab} \in \mathcal{X}$ and :

$$P_S(a, b) = \mathrm{in}(c'_S, < a, b >).\, \mathrm{new}\, K_{ab}.\, \mathrm{out}(c'_S, \mathrm{enc}(<< b, K_{ab} >, \mathrm{enc}(< K_{ab}, a >, \mathrm{shk}(b)) >, \mathrm{shk}(a)))$$

where $c_S \in Ch_0$ and K_{ab} is a name.

The process $P_{DS} = P_A(a, b)|P_B(a, b)|P_S(a, b)$ is a protocol, but not a restricted one.

2.3 Semantics

As we want to define a semantics for the phase, we say that $t : P$ is a *phased process* if t is a natural integer and P a process. It is ground if P is ground.

Before defining operationnal semantics, we need to define the configurations, because the semantics is a relation between configurations. A *configuration* is a triple $(\mathcal{P}, \phi, t)$ where :

— $\mathcal{P}$ is a finite multiset of ground phased processes. The phase of the processes represent the phase in which they may be used.
— ϕ is a substitution $\{w_1 \triangleright m_1, \ldots, w_n \triangleright m_n\}$ where each w_i is a variable of $\mathcal{W}$ and each m_i is a message. We say that ϕ is a *frame*. The messages m_i represent the knowledge of the attacker.
— t is a natural integer. It is used to represent the current phase. We denote phase$(C) = t$.

Sometimes, I write P instead of $\{0 : P\}$ or instead of $(0 : P, \emptyset, 0)$. I also write $t : P$ instead of $\{t : P\}$. Moreover, we can see substitutions, and in particular frames, as sets of rules $w \triangleright t$. In particular, it is possible to write $\phi_1 \cap \phi_2$ the frame such that $dom(\phi_1 \cap \phi_2) = \{w \in dom(\phi_1) \cap dom(\phi_2) | w\phi_1 = w\phi_2\}$ and $\forall w \in dom(\phi_1 \cap \phi_2), w(\phi_1 \cap \phi_2) = w\phi_1 = w\phi_2$.

We say that a configuration is a *restricted* configuration if its phased processes are restricted processes and if its frame only refers to terms without errors.

The semantics is given by the following rules :

$[Null]$ $(t : \{0\} \cup \mathcal{P}, \phi, t) \xrightarrow{\tau} (\mathcal{P}, \phi, t)$

$[Sess]$ $(t : !^{c}_{c'} P \cup \mathcal{P}, \phi, t) \xrightarrow{\text{sess}(c, ch_i)} (t : P\{^{ch_i}/_{c'}\} \cup t : !^{c}_{c'} P \cup \mathcal{P}, \phi, t)$
with ch_i a new fresh name : $ch_i \in Ch^{fresh}$.

$[Par]$ $(t : (P|Q) \cup \mathcal{P}, \phi, t) \xrightarrow{\tau} (t : P \cup t : Q \cup \mathcal{P}, \phi, t)$

$[New]$ $(t : \text{new}\, n.P \cup \mathcal{P}, \phi, t) \xrightarrow{\tau} (t : P\{^{n'}/_{n}\} \cup \mathcal{P}, \phi, t)$
with n' a new fresh name in $\mathcal{N}$.

$[Input]$ $(t : \text{in}(c, u).P \cup \mathcal{P}, \phi, t) \xrightarrow{\text{in}(c,R)} (t : P\sigma \cup \mathcal{P}, \phi, t)$
where R is a recipe and when $R\phi\downarrow$ is a message unifiable with u with most general unifier σ.

$[Const]$ $(t : \text{out}(c, a).P \cup \mathcal{P}, \phi, t) \xrightarrow{\text{out}(c,a)} (t : P \cup \mathcal{P}, \phi, t)$ with $a \in \Sigma_0^{pub}$.

$[Out]$ $(t : \text{out}(c, u).P \cup \mathcal{P}, \phi, t) \xrightarrow{\text{out}(c,w)} (t : P \cup \mathcal{P}, \phi \cup \{w \triangleright u\}, t)$
when $u \in \mathcal{M}_\Sigma \backslash \Sigma_0^{pub}$ where w is a new variable of $\mathcal{W}$.

$[Pass]$ $(t : \text{let}\, x = v \,\text{in}\, P \,\text{else}\, E \cup \mathcal{P}, \phi, t) \xrightarrow{\tau} (t : P\{^{v\downarrow}/_{x}\} \cup \mathcal{P}, \phi, t)$ when $v\downarrow \in \mathcal{M}_\Sigma$.

$[Fail]$ $(t : \text{let}\, x = v \,\text{in}\, P \,\text{else}\, E \cup \mathcal{P}, \phi, t) \xrightarrow{\tau} (t : E \cup \mathcal{P}, \phi, t)$ when $v\downarrow \notin \mathcal{M}_\Sigma$.

$[Plan]$ $(t : \text{phase}\, t'.P \cup \mathcal{P}, \phi, t) \xrightarrow{\tau} (t' : P \cup \mathcal{P}, \phi, t)$ where t' is an integer.

$[Phase]$ $(\mathcal{P}, \phi, t) \xrightarrow{\text{phase}\, t'} (\mathcal{P}, \phi, t')$ with $t' > t$.

$[Clean]$ $(t : P \cup \mathcal{P}, \phi, t') \xrightarrow{\tau} (\mathcal{P}, \phi, t')$ when $t' > t$.

The rules correspond to the intuitive meaning of the syntax given above. The rule [Null] is the elimination of the null process. The rule [Sess] allows the attacker to replicate a process that operates on a new fresh channel. The rule [Par] allows the attacker to use both P and Q when $(P|Q)$ is available. The rule [Input] allows the attacker to input a message in a process expecting it. The rule [Const] allows the attacker to get the public constant a to be outputed. The rule [Out] allows the attacker to get a message from a protocol : this message is added in the attacker's knowledge. The rule [Pass] allows to pass into P when v reduces into a message (and instantiate x with v in P). The rule [Fail] allows to pass into the error branch E when $v\downarrow$ doesn't reduces into a message. The rule [Plan] gives the process

the phase in which it can continue to be executed. The rule [Phase] allows to increase the phase. The rule [Clean] allows to eliminate all processes that stay in former phases (as phase can only increase, they can't be executed anymore). We assume that changing the phase is observable, so the observable actions are $\text{in}, \text{out}, \text{sess}, \text{phase}$.

The relation $\xrightarrow{\alpha}$ between configurations defined by the rules above can be extended into its transitive closure $\xrightarrow{\alpha_1 \dots \alpha_n}$. If tr is α where all the τ-actions have been removed, then we write $C \overset{\text{tr}}{\Longrightarrow} C'$ when $C \xrightarrow{\alpha} C'$ (and C, C' are configurations). The set $trace(C) = \{(\text{tr}, \phi) | C \overset{\text{tr}}{\Longrightarrow} (\mathcal{P}, \phi, t)$ for some multiset $\mathcal{P}$ and some integer $t\}$ is the set of traces of the configuration C. We define the bounded variables of a trace tr as the variables that occur in the (non-error) outputs of tr, and we denote them $bv(\text{tr}) \subset \mathcal{W}$. If $(\mathcal{P}, \phi, t) \overset{\text{tr}}{\Longrightarrow} (\mathcal{P}', \phi \cup \psi, t')$ it is obvious that $dom(\psi) = bv(\text{tr})$.

Example 8 : Take the process P_{DS} of example 7. Define :

$$\text{tr} = \text{out}(c'_A, w_1).\,\text{in}(c'_S, w_1).\,\text{out}(c'_S, w_2).\,\text{in}(c'_A, w_2).\,\text{out}(c'_A, w_3).\,\text{in}(c'_B, w_3)$$

tr is the trace of an honest execution of $(0 : P_{DS}, \emptyset, 0)$, that is an execution where the messages have been transmitted according to the specification. A resulting frame could be :

$$\phi_0 = \{w_1 \triangleright < ag_1^H, ag_2^H >; w_2 \triangleright \text{enc}(<< ag_2^H, k >, \text{enc}(< k, ag_1^H >, Kbs) >, Kas);$$
$$w_3 \triangleright \text{enc}(< k, ag_1^H >, Kbs)\}$$

where k is a name ($k \in \mathcal{N}$). And so a possible execution of this trace is :

$$(0 : P_{DS}, \emptyset, 0) \overset{\text{tr}}{\Longrightarrow} (\emptyset, \phi_0, 0)$$

2.4 Trace equivalence

Trace equivalence is a way to state indistinguishability from the point of view of the attacker. It is very useful to formalize privacy properties. Before defining the trace equivalence, we need to define static equivalence, that is indistinguishability between frames from the point of view of the attacker.

Definition 3 : *Let ϕ and ψ be two frames. We say that ϕ is* statically included *into ψ, and we denote $\phi \sqsubset \psi$, if the three following properties are true :*

- *$dom(\phi) \subset dom(\psi)$*
- *For all recipe $R \in \mathcal{T}(\Sigma_{pub}, \mathcal{W})$, if $R\phi\downarrow \in \mathcal{M}_\Sigma$ then $R\psi\downarrow \in \mathcal{M}_\Sigma$.*
- *For all recipes $R_1, R_2 \in \mathcal{T}(\Sigma_{pub}, \mathcal{W})$, such that $R_1\phi\downarrow$ and $R_2\phi\downarrow$ are messages, we have that $(R_1 = R_2)\phi$ implies $(R_1 = R_2)\psi$.*

We say that ϕ and ψ are in static equivalence*, and we denote $\phi \sim \psi$, if $\phi \sqsubset \psi$ and $\psi \sqsubset \phi$. Sometimes, we will also denote $C_P \sim C_Q$ or $C_P \not\sim C_Q$ when the frames of configurations C_P and C_Q are not in static equivalence.*

Example 9 : Assume that m_0 and m_1 are public constants. We define :

$$\phi = \{w_4 \triangleright \text{enc}(m_0, k)\,;\, w_5 \triangleright \text{enc}(m_0, k)\}$$

$$\psi = \{w_4 \triangleright \text{enc}(m_1, n)\,;\, w_5 \triangleright \text{enc}(m_0, n')\}$$

We have $\phi \not\sim \psi$ because $w_4\phi\downarrow = w_5\phi\downarrow$ but $w_4\psi\downarrow \neq w_5\psi\downarrow$.

We can now define the notion of trace equivalence. Two protocols stay in trace equivalence if they stay equivalent whatever trace is executed. That is, after every execution, it is impossible for the attacker to distinguish between the two frames that may have resulted from the execution of the process.

Definition 4 : *Let C and C' be two configurations. We have that C is* trace-included *in C', and we denote $C \sqsubseteq C'$, if for every $(\mathrm{tr}, \phi) \in trace(C)$, there exists ϕ' such that $(\mathrm{tr}, \phi') \in trace(C')$ and $\phi \sim \phi'$. We say that C and C' are in* trace equivalence*, and we denote $C \approx C'$ if $C \sqsubseteq C'$ and $C \sqsubseteq C'$.*

We say that tr *is a* trace of non-inclusion *$C \not\sqsubseteq C'$ if there exists ϕ such that $(\mathrm{tr}, \phi) \in trace(C)$ but there exists no ϕ' such that $(\mathrm{tr}, \phi') \in trace(C')$ and $\phi \sim \phi'$. We say that* tr *is a* trace of non-equivalence *$C \not\approx C'$ if it is a trace of non-inclusion $C \not\sqsubseteq C'$ or $C' \not\sqsubseteq C$.*

Note that as I write sometimes P for $(\{0 : P\}, \emptyset, 0)$, I may write $P \approx Q$ instead of $(\{0 : P\}, \emptyset, 0) \approx (\{0 : Q\}, \emptyset, 0)$.

Note that if phase$(C) \neq$ phase(C'), then $C \not\approx C'$. Indeed, assume wlog that $t =$ phase$(C) <$ phase(C'). Then phase$(t + 1)$ is a trace of C but is not a trace of C'. So we have that $C \approx C'$ iff $C \approx C'$ and phase$(C) =$ phase(C').

Example 10 : Consider the protocols P_A and P_S of Example 7. We define P'_B and Q'_B as :

$$P'_B(a,b) = !^{c_B}_{c'_B} P_B(a,b).\,\mathrm{out}(c'_B, \{m_0\}_{z_{ab}})$$
$$Q'_B(a,b) = !^{c_B}_{c'_B} P_B(a,b).\,\mathrm{new}\,k.\,\mathrm{out}(c'_B, \{m_1\}_k)$$

where m_0, m_1 are public constants. The property we have encoded here is a strong secrecy variant introduced in [7]. Then define :

$$P = P_A(a,b) | P'_B(a,b) | P_S(a,b)$$
$$Q = P_A(a,b) | Q'_B(a,b) | P_S(a,b)$$

We consider the trace :

$$\mathrm{tr} = \mathrm{out}(c'_A, w_1).\,\mathrm{in}(c'_S, w_1).\,\mathrm{out}(c'_S, w_2).\,\mathrm{in}(c'_A, w_2).\,\mathrm{out}(c'_A, w_3).$$
$$\mathrm{sess}(c_B, ch_1).\,\mathrm{in}(ch_1, w_3).\,\mathrm{out}(ch_1, w_4).\,\mathrm{sess}(c_B, ch_2).\,\mathrm{in}(ch_2, w_3).\,\mathrm{out}(ch_2, w_5)$$

tr is a trace of both $C_P = (0 : P, \emptyset, 0)$ and $C_Q = (0 : Q, \emptyset, 0)$. We have that $(\mathrm{tr}, \phi_0 \cup \phi) \in trace(C_P)$ and $(\mathrm{tr}, \phi_0 \cup \psi) \in trace(C_Q)$ where ϕ_0 has been defined in Example 8 and ϕ and ψ have been defined in Example 9. We have that $\phi_0 \cup \phi \not\sim \phi_0 \cup \psi$ and P and Q are not in trace equivalence.

The trace tr is the trace of a real attack : once the attacker has observed an honest execution, he can send once more the last message to the attacker and get him to reuse the same key. Real implementations of Denning Sacco protocols add time stamps to avoid this attack.

3 Contribution

The goal is to bound the number of agents involved in protocols P and Q to study their equivalence. Before doing that, we need to define a model for the protocol P using n_H honest agents and n_D

dishonest ones : so, we will add the names of the agents, and some related informations (e.g. their public or leaked keys) to the attacker's knowledge, that is in initial frames ϕ_k where $n_H, n_D \leq k$. Then, we will have to prove that there exists k_0 such that for any $k \geq 0$, $(P, \phi_k, 0) \approx (Q, \phi_k, 0)$ iff $(P, \phi_{k_0}, 0) \approx (Q, \phi_{k_0}, 0)$ which will then allow to verify only equivalences with a bounded number of agents. Finally I consider some extensions and I explain why they are not possible (see section 6.1).

To write the initial knowledge of the attacker, we assume that there are private symbols $\mathrm{ag}/1$, $\mathrm{hon}/1$, $\mathrm{dis}/1$, that are used to state an information on the agents. $\mathrm{ag}(x)$ symbolises the fact that x is the name of an agent. hon is used to represent honest agents, and dis is used to represent dishonest agents. We sometimes have to require for a process to be executed with an honest agent only because attack can only be done against honest session. We also need the predicate dis to give the attacker the informations he needs about dishonest agents, like their shared keys. To do that, we will add a dishonest key distribution protocol.

We assume also that there is a public pair function symbol $<,>/2$, two public destructor symbols proj_1 and proj_2, and a rewriting rule $\mathrm{proj}_i(< x_1, x_2 >) \longrightarrow x_i$ for each i. To simplify, we will use a public tuple constructor symbol $<, \ldots, >$ together with associated projection, but note that it can be encoded with the pair and the two projection symbols above. So, let n_H, n_D be two natural integers. We define :

$$\begin{aligned}\phi_k = \{&w_1^H \triangleright < ag_H^1, \mathrm{ag}(ag_H^1), \mathrm{hon}(ag_H^1) >; \ldots;\\ &w_k^H \triangleright < ag_H^k, \mathrm{ag}(ag_H^k), \mathrm{hon}(ag_H^k) >;\\ &w_1^D \triangleright < ag_D^1, \mathrm{ag}(ag_D^1), \mathrm{dis}(ag_D^1) >; \ldots;\\ &w_k^D \triangleright < ag_D^k, \mathrm{ag}(ag_D^k), \mathrm{dis}(ag_D^k) >\}\end{aligned}$$

Note that if we want to add some constant agent (e.g. a server s) we can for exemple add a :

$$w_S \triangleright < s, \mathrm{ag}(s), \mathrm{hon}(s) >$$

to the attacker's knowledge.

For these frames to be useful, we need that the names in the specification of a protocol do not refer to the names of agents : protocols are instanciated later, by the attacker, through a prefix in the protocols. Indeed, for a protocol between two agents, we need to add a $\mathrm{in}(c, < \mathrm{ag}(x), \mathrm{ag}(y) >)$ at the beginning. The attacker can make an input $\mathrm{in}(c, < \mathrm{ag}(ag_H^1), \mathrm{ag}(ag_H^2) >)$ and the process is then instanciated as the protocol between ag_H^1 and ag_H^2. Consider our running example :

Example 11 : If we want to use an unbounded number of agents (and of sessions), we have to extend our signature :

$$\Sigma = \{\mathrm{ag}/1; \mathrm{hon}/1; \mathrm{dis}/1; \mathrm{shk}/1; \mathrm{enc}/2; \mathrm{dec}/2; MacError/0; <,>/2; \mathrm{proj}_1/1; \mathrm{proj}_2/1\} \cup (\Sigma_0^{pub})^-$$

where $\mathrm{ag}, \mathrm{hon}, \mathrm{dis}$ and shk are private constructor terms (other symbols and the rules are as in Example 7).

We redefine our processes :

$$P'_A = !^{c_A}_{c'_A} \text{in}(c'_A, < \text{ag}(x_1), \text{ag}(x_2) >).P_A(x_1, x_2)$$

$$P^H_B = !^{c^H_B}_{c'_B} \text{in}(c'_B, < \text{hon}(y_1), \text{hon}(y_2) >).P_B(y_1, y_2).\,\text{phase}\,1.\,\text{out}(c'_B, \{m_0\}_{z_{ab}})$$

$$Q^H_B = !^{c^H_B}_{c'_B} \text{in}(c'_B, < \text{hon}(y_1), \text{hon}(y_2) >).P_B(y_1, y_2).\,\text{phase}\,1.\,\text{new}\,k.\,\text{out}(c'_B, \{m_1\}_k)$$

$$P^D_B = !^{c^D_B}_{c'_B} \text{in}(c'_B, < \text{ag}(y'_1), \text{ag}(y'_2) >).P_B(y'_1, y'_2)$$

$$P'_S = !^{c_S}_{c'_S} \text{in}(c'_S, < \text{ag}(z_1), \text{ag}(z_2) >).P_S(z_1, z_2)$$

Then we define $P_0 = P'_A|P^H_B|P^D_B|P'_S$ and $Q_0 = P_A|Q^H_B|P^D_B|P'_S$. P_0 and Q_0 are both restricted protocols (there is no name of agent in their terms). So it is up to the attacker to instanciate the protocols by the agents names.

Note that the attacker should be able to get the keys of the dishonest agents. So either we add these keys in the initial frame ϕ_k, or we add a protocol that reveal these keys. In this example, we need the following dishonest key revealation protocol :

$$K = \\ !^{c_K}_{c'_K} \text{in}(c'_K, \text{dis}(x)).\,\text{out}(c'_K, \text{shk}(x))$$

Note that the secret keys of the dishonest agents (for asymmetric encryption) can also be handled this way. Finally, the protocols are $P = P_0|K$ and $Q = Q_0|K$.

We will prove our result in several parts. First of all, we will prove that when protocols are equivalents with a lot of agents, they are still equivalent with a smaller number of agents (lemma 1). Then, we will prove that there is some class of protocols and some integer k_0 such that $(P, \phi_{k_0}, 0) \approx (Q, \phi_{k_0}, 0)$ implies $\forall k \geq k_0, (P, \phi_k, 0) \approx (Q, \phi_k, 0)$. This will allow us to prove that $\forall k, (P, \phi_k, 0) \approx (Q, \phi_k, 0)$ iff $(P, \phi_{k_0}, 0) \approx (Q, \phi_{k_0}, 0)$.

4 Action-determinism

In this section, we present our main hypothesis, called action-determinism, and two consequences of it.

We say that a protocol is *simple* if it is written in the following form :

$$P = !^{c'_1}_{c_1} B_1 \mid \ldots \mid !^{c'_m}_{c_m} B_m \mid B_{m+1} \mid B_{m+p}$$

where each B_i is a basic processes built on channel c_i, that is processes written under the following form :

$$B_i = 0 \mid \text{in}(c_i, u).P \mid \text{out}(c_i, u).P \mid \text{new}\, n.P \mid \text{let}\, y = v\, \text{in}\, P\, \text{else}\, E \mid \text{phase}\, t.P$$

Note that the protocols written in our examples were all simple protocols.

The main hypothesis for our result is that each action of a given trace can only be used on one process. More formally :

Definition 5 : *Let C be a configuration. We say that C is action-deterministic if $\forall \text{tr}, \mathcal{P}, \phi, t$ such that $P \xRightarrow{\text{tr}} (\mathcal{P}, \phi, t)$, $\forall t : P, t : Q \in \mathcal{P}$, if $P = \alpha.P'$ and $Q = \beta.Q'$ then α and β are either not of the same nature (in, out or $!^c_{c'}$) or don't occur on the same channel. By extension, we say that a protocol P is action-deterministic if $(0 : P, \phi, t)$ is action-deterministic for any ϕ, t.*

Note that when $C \xRightarrow{\text{tr}} C'$ with C action-deterministic, then C' is action-deterministic. Moreover, if P is a simple protocol, then it is obvious that P is action-deterministic (at any time, only one action is reachable on a given channel because basic processes are sequences of actions, and each basic process works on its own channel).

Most real protocols are action-deterministic because the attacker may often identify processes and session through session identifiers or IP adresses. Moreover, specifying a protocol as an action-deterministic process gives more power to the attacker and can be seen as good practice.

For action-deterministic restricted protocols, we can assume that at any point, there are no error outside else branches, and that in a trace of non-equivalence, the only occurence of an output of error is in the last action.

Property 1 : *Let C_P and C_Q be two action-deterministic restricted configurations. If $C_P \not\approx C_Q$, then there exists a trace of non-equivalence* tr *such that there is no error in input or output in* tr*, except perhaps in one* $\text{out}(c, e)$ *at the end (that is, if there is an error e occuring in* tr*, then* $\text{tr} = \text{tr}' \,.\, \text{out}(c, e)$ *for some channel c where there is no error in* tr'*).*

Sketch of proof. Any error constant injected in input may be replaced by a non-error public constant in input without modification of the execution, because restricted protocols do not have errors outside their else branches. So as errors can only appear in the protocol through inputs, we may assume that at any step of the execution, there is no error outside else branches. So any $\text{out}(c, e)$ occuring in a trace of non-equivalence may be assumed to refer to an output in an elsse branch reached by the $[Fail]$ rule. But our small else branches do not help to reach other processes, so they may all be executed at the end. As our protocols are ation-deterministic, there is only one such reachable $\text{out}(c, e)$ at the end, which allows us to conclude.

Moreover, in action-deterministic processes, there is only one way to execute each trace : that is, once the trace tr is fixed, there is a unique frame reachable through tr. More formally :

Property 2 : *Let P be an action-deterministic protocol. Let* tr *be a trace, let $(\mathcal{P}_1, \phi_1, t_1)$ and $(\mathcal{P}_2, \phi_2, t_2)$ such that $P \Longrightarrow^{\text{tr}} (\mathcal{P}_1, \phi_1, t)$ and $P \Longrightarrow^{\text{tr}} (\mathcal{P}_2, \phi_2, t')$.*

Then $\phi_1 = \phi_2$ (up to alpha-renaming).

Sketch of proof. We first show that silent actions can be executed asap without modifying the resulting frame. This gives us a unique way to execute a fixed trace tr and thus the result.

5 Unbounded number of agents

5.1 First steps

The easy part of the theorem is to prove that when protocols are equivalent with th respect to executions that may involve up to k agents,then they are equivalent with respect to executions that involve fewer agents. More formally, we have the following lemma :

Lemma 1 : *Let P and Q be two restricted protocols. Let $k \leq k'$ be two integers. If $(P, \phi_{k'}, 0) \approx (Q, \phi_{k'}, 0)$ then $(P, \phi_k, 0) \approx (Q, \phi_k, 0)$.*

For any pair of integers $k \geq k_0$, we will consider some special set of renamings $\mathcal{F}^k_{k_0} = \{\rho \mid dom(\rho) \subset \mathcal{A}_k, \mathcal{A}^H_k \rho \subset \mathcal{A}^H_{k_0}, \mathcal{A}^D_k \rho \subset \mathcal{A}^D_{k_0}\}$. Intuitively, $\mathcal{F}^k_{k_0}$ is a set of renaming that rename only the k first agents, and that send them on the k_0 first.

Now, after lemma 1, we only have to prove that there is a k_0 such that $(P, \phi_{k_0}, 0) \approx (Q, \phi_{k_0}, 0)$ implies $\forall k \geq k_0, (P, \phi_k, 0) \approx (Q, \phi_k, 0)$. To do that, we consider the following properties :

1. $(P, \phi_{k_0}, 0) \approx (Q, \phi_{k_0}, 0)$
2. $\forall k \geq k_0, \forall \rho \in \mathcal{F}^k_{k_0}, (P, \phi_k \rho, 0) \approx (Q, \phi_k \rho, 0)$
3. $\forall k \geq k_0, (P, \phi_k, 0) \approx (Q, \phi_k, 0)$

To prove our theorem, it is sufficient to prove that (1) $\Rightarrow$ (3). The second one will be an intermediate step that allows us to handle frame and trace of non-equivalence separately : the modification of the trace is done by (1) $\Rightarrow$ (2) and the modification of the frame is done by (2) $\Rightarrow$ (3). Note that we will actually get the equivalence between the three properties as (3) $\Rightarrow$ (1) is obvious.

Now, we consider the implication (1) $\Rightarrow$ (2), which is given by the following lemma :

Lemma 2 : *Let $k \geq k_0$ be integers. Let $\rho \in \mathcal{F}^k_{k_0}$. If $(P, \phi_{k_0}, 0) \approx (Q, \phi_{k_0}, 0)$ then $(P, \phi_k \rho, 0) \approx (Q, \phi_k \rho, 0)$.*

Sketch of proof : By definition of $\mathcal{F}^k_{k_0}$, $\phi_k \rho$ is a frame where at most k_0 different names of honest agents and k_0 different names of dishonest agents appear. So we can assume that the names of agents occuring in $\phi_k \rho$ are those of ϕ_{k_0} : the attacker's knowledge resulting from the frames is actually the same. So the reachable configurations are the same with almost the same trace (it is sufficient to change variables $w \in dom(\phi_k)$ into variables $w' \in dom(\phi_{k_0})$ such that $w\phi_k\rho\downarrow = w'\phi_{k_0}\downarrow$). As traces of non-equivalence with $\phi_k \rho$ may be transformed into traces of non-equivalence with ϕ_{k_0}, the result is true.

5.2 Bound

The bound k_0 depends on the equational theory through a number that indicates how many names have to be blocked to avoid a term to reduce. More formally, let ρ be a renaming and n a name. We say that ρ is *n-adequat* if $n\rho = n$ and $\forall n' \neq n, n'\rho \neq n$. If N is a set of names, we say that ρ is *N-adequat* if it is n-adequat for any $n \in N$. We say that a theory is $\mathfrak{b}$-blockable if it is possible to block any reduction with $\mathfrak{b}$ names. Formally :

Definition 6 : *Let $(\Sigma, \mathcal{R})$ be a theory. Let $\mathcal{M}_\Sigma \subset \mathcal{T}(\Sigma_c, \hat{\mathcal{N}})$ be a notion of messages. We say that $(\Sigma, \mathcal{R})$ is $\mathfrak{b}-$blockable w.r.t. $\mathcal{M}_\Sigma$ if for any $t \in \mathcal{T}(\Sigma, \hat{\mathcal{N}}) \backslash \mathcal{M}_\Sigma$ in normal form, there exists a set of names $N \subset \hat{\mathcal{N}}$ of size at most $\mathfrak{b}$ such that for any N-adequat renaming ρ, $t\rho\downarrow \notin \mathcal{M}_\Sigma$. We say that $\mathfrak{b}$ the blocking number of $(\Sigma, \mathcal{R})$ w.r.t. $\mathcal{M}_\Sigma$ if it is the smallest number such that the theory is $\mathfrak{b}$-blockable.*

Any theory with a finite set of rewriting rules is $\mathfrak{b}$-blockable for some $\mathfrak{b}$ and a bound on the blocking number can be computed :

Property 3 : *Let $(\Sigma, \mathcal{R})$ be a theory with $\mathcal{R}$ finite. Let $\mathcal{M}_\Sigma \in \mathcal{T}(\Sigma_c, \hat{\mathcal{N}})$ be a notion of messages. Let $\mathfrak{b}$ be the maximal number of rules that use the same destructor. Then $(\Sigma, \mathcal{R})$ is $\mathfrak{b}$-blockable w.r.t. $\mathcal{M}_\Sigma$.*

This property implies that the theory of our running example is 1-blockable. Actually, it is quite rare that some destructor may be reduced by several rules, and 1-blockable theories are sufficient to modelize standard cryptographic primitives like symmetric and asymmetric encryption, signature, pair, and hash.

Example 12 : Let $\Sigma_c = \{<,> /2, ok/0\}$, and $\Sigma_d = \{\mathrm{eq}_3 /6\}$. Let $\mathcal{R}$ be the following set of rewriting rules :

$$\mathcal{R} = \{\, \mathrm{eq}_3(x,x,y_1,y_2,y_3,y_4) \to ok\,;\, \mathrm{eq}_3(y_1,y_2,x,x,y_3,y_4) \to ok\,;\, \mathrm{eq}_3(y_1,y_2,y_3,y_4,x,x) \to ok\,\}$$

The theory $(\Sigma, \mathcal{R})$ is convergent and its blocking number is 3 after property 3 because the 3 rules have the same destructor at top level position. E.g. the term $eq(n_1, n_1', n_2, n_2', n_3, n_3')\rho$ does not reduce for any N-adequat renaming ρ with $N = \{n_1, n_2, n_3\}$, but it is not possible to choose such a set N of size 2.

This example can be generalized with a symbol $eq_n/2n$, but the critical number will increase to n.

5.3 Main result

The lemma 3 is the most difficult part of the theorem. It allows us to prove (2) $\Rightarrow$ (3), and to get our main theorem :

Lemma 3 : *Let $(\Sigma, \mathcal{R})$ be any finite convergent theory and $\mathcal{M}_\Sigma$ be a notion of messages such that $(\Sigma, \mathcal{R})$ is $\mathfrak{b}$ blockable w.r.t. $\mathcal{M}_\Sigma$, and denote $k_0 = 2\mathfrak{b} + 1$. Let $k \geq k_0$ be an integer. Let P and Q be two action-deterministic restricted protocols. Then :*

$$[\forall \rho \in \mathcal{F}^k_{k_0}.(P, \phi_k\rho, 0) \approx (Q, \phi_k\rho, 0)] \Rightarrow (P, \phi_k, 0) \approx (Q, \phi_k, 0)$$

Sketch of proof. We consider a trace of non-equivalence tr and we assume it is a trace of non-inclusion $P \not\sqsubseteq Q$. Thanks to property 1, we assume that no error occurs in tr except perhaps in the last action. Then tr passes in P so it still passes in P after any renaming (because renaming preserves equality and so succeeding tests) except if the last action is an output of error : in this case, we need to preserve at most $\mathfrak{b}$ names to make the final test fail. Regarding the Q side, we have to preserve a failing let (that is the fact that a $[Fail]$ rule is used instead of a $[Pass]$ one), or a non-reducing term, or a disequality. Each of them can be preserved by keeping at most $\mathfrak{b}$ names. So we have at most $2\mathfrak{b}$ names (of honest and of dishonest agents) to preserve, and we need 2 : one to project all honest agents, and one to project all dishonest agents. It implies that we need at most $2\mathfrak{b} + 1$ names of each kind (honest or dishonest) of agents.

Now, we are ready to prove our main theorem :

Theorem 1 : *So let $(\Sigma, \mathcal{R})$ be a $\mathfrak{b}$-blockable convergent theory w.r.t. the notion of messages $\mathcal{M}_\Sigma$. Denote $k_0 = 2\mathfrak{b} + 1$. Let $k \geq k_0$ be an integer. Let P and Q be two action-deterministic restricted protocols. Then $\forall k \in \mathbb{N}, (P, \phi_k, 0) \approx (Q, \phi_k, 0)$ iff $(P, \phi_{k_0}, 0) \approx (Q, \phi_{k_0}, 0)$.*

Proof : Thanks to lemma 1, it is sufficient to prove $(P, \phi_{k_0}, 0) \approx (Q, \phi_{k_0}, 0) \Rightarrow \forall k \geq k_0, (P, \phi_k, 0) \approx (Q, \phi_k, 0)$.

But lemma 2 gives us that $(P, \phi_{k_0}, 0) \approx (Q, \phi_{k_0}, 0) \Rightarrow \forall k \geq k_0, \forall \rho \in \mathcal{F}^k_{k_0}, (P, \phi_k\rho, 0) \approx (Q, \phi_k\rho, 0)$, and $\forall k \geq k_0, \forall \rho \in \mathcal{F}^k_{k_0}, (P, \phi_k\rho, 0) \approx (Q, \phi_k\rho, 0) \Rightarrow \forall k \geq k_0, (P, \phi_k, 0) \approx (Q, \phi_k, 0)$ by lemma 3.

6 Scope of the result

6.1 Counter examples

We have assumed that our processes have only else branches of a certain kind, and it may seem arbitrary, as for our action-determinism hypothesis, and the form of our rules (that is, destructors on constructor terms). However, these hypothesis are necessary in the sense that it is not possible to extend our result to processes with non-restricted else branches, to pure equational theories or to non-action deterministic protocols.

Our examples rely on non-computability of a bound on the size of a solution of the Post Correspondance Problem (PCP). We use a classic example for undecidability, and we add a list of agents that increases with the size of the solution. Then, at the end, we find a way to do that the protocols are not equivalent iff the agents of this list are pairwise distinct. In this section, I explain the counter example with general else branches. For a more formal description of this example and the others, see appendix B.

The first step is to encode the PCP problem with our list (represented by z_ℓ and $< \mathrm{ag}(x_{new}), z_\ell >$). At the end, the two protocols will be different after verification that all agents were distincts. Consider the alphabet A and let $(u_i, v_i)_{1 \leq i \leq n} \in A^\star$ be an instance of PCP. We consider the following protocols specified intuitively. They can easily be encoded as action deterministic processes.

The following process allows the attacker to build inductively a solution of PCP. The first line gives an first tile and the others allow to build bigger words.

$$
\begin{aligned}
&P_{PCP} = \\
&\to \mathrm{enc}(<< u_1, v_1 >, end >, k_{PCP}) \\
&\mathrm{ag}(x_{new}), \mathrm{enc}(<< x, y >, \ell >, k_{PCP}) \to \mathrm{enc}(<< xu_1, yv_1 >, < \mathrm{ag}(x_{new}), z_\ell >>, k_{PCP}) \\
&\dots \\
&\mathrm{ag}(x_{new}), \mathrm{enc}(<< x, y >, \ell >, k_{PCP}) \to \mathrm{enc}(<< xu_n, yv_n >, < \mathrm{ag}(x_{new}), z_\ell >>, k_{PCP})
\end{aligned}
$$

The following protocols check whether a solution of the PCP instance has been reached and that the list x_ℓ has been approved (*yes* and *no* are public constants).

$$
\begin{aligned}
&P_{check} = \\
&< \mathrm{enc}(<< x, x >, x_\ell >, k_{PCP}), \mathrm{enc}(x_\ell, k_{approved}) > \to yes
\end{aligned}
$$

$$Q_{check} = \\ < \mathrm{enc}(<< x, x >, x_\ell >, k_{PCP}), \mathrm{enc}(x_\ell, k_{approved}) > \rightarrow no$$

Now, we have to ensure that the list x_ℓ is approved iff its elements are pairwise distinct. A list (represented as a pair of a head element and a tail list) can be approved iff it only contains pairwise distinct agent names. P_{init} is used to allow any list containing only one element.

$$P_{init} = \\ < \mathrm{ag}(x), end > \rightarrow \mathrm{enc}(< \mathrm{ag}(x), end >, k_{approved})$$

Then, P_{diff} is used to check that the elements of a pair are distinct.

$$P_{diff} = \\ < \mathrm{ag}(x), \mathrm{ag}(y) > \rightarrow [\mathrm{let}\, z = \mathrm{eq}(x, y)\, \mathrm{in}\, 0\, \mathrm{else}\, \mathrm{enc}(< x, y >, k_{diff})]$$

Now, $P_{decompose}$ is used to get bigger lists. That is, when a list $< x, y >$ and $< x', y >$ are approved, then $< x, < x', y >>$ is also approved if $x \neq x'$. Indeed, a pair of elements of $< x, < x', y >>$ may be a pair of elements of y, or x and an element of y, or x' and an element of y, or the pair $< x, x' >$.

$$P_{decompose} = < \mathrm{enc}(< x, x' >, k_{diff}), \mathrm{enc}(< x, y >, k_{approved}), \mathrm{enc}(< x', y >, k_{approved}) > \\ \rightarrow \mathrm{enc}(< x, < x', y >>, k_{approved})$$

Then it is obvious that we can only get $\mathrm{enc}(x, k_{approved})$ if we know x_0 and x is a sublist of x_0 containing only agent names with elements pairwise distinct. The only part that remains to prove is that we may have to use it for arbitrary long lists of public keys.

Now define (with the ! notation of pi-calculus) :

$$P = !P_{init} | !P_{diff} | !P_{main} | !P_{PCP} | !P_{check}$$

$$Q = !P_{init} | !P_{diff} | !P_{main} | !P_{PCP} | !Q_{check}$$

These encoding can be made more realistic at the price of less readability. Indeed, the variables $\mathrm{ag}(x)$ can be written all at the beginning, or the protocol P_{diff} can be seen as a server signing $< x, y >$. With some more care, these protocols can be expressed as action-deterministic processes (see appendix B.1).

There is an attack when the PCP instance has a solution, and the list involved in this attack is of the same size (in number of tiles) as the solution. But no bound on the size of the PCP attack is computable, so the number of agents involved in an attack is not computable.

Other counter examples : The counter-example with pure equational theories relies on the fact that with pure equationnal theories, it is possible to build inductively two big terms t_P and t_Q that will be different iff the elements of some list of arbirary size are pairise distinct. The counter example with non-action-deterministic protocols is based on the same idea as the one with else branches except that non-determinism is used instead of else branches. In this setting, the protocols P and Q can be used deterministically if there are enough agents, but they become non-deterministic if they are used with agents that are not pairwise distinct.

These three counter examples show that our hypothesis are natural. Note that if we cannot use general else branches, the error branches and the $\mathrm{eq}(x_1, \ldots, x_{2n})$ constructions allow us to handle a large class of else branches.

6.2 Unlinkability

We inherit limits of our action-determinism hypothesis : most of the unlinkability encodings are not possible, because there is not the same number of replications both sides. Consider for example the passport protocol. The encoding of the unlinkability property is as follows :

$$!\,\mathrm{new}\, k_e.\,\mathrm{new}\, k_m.!(P|R) \approx !\,\mathrm{new}\, k_e\,\mathrm{new}\, k_m(P|R)$$

where P is the protocol played by the passport and R the protocol played by the reader. The left handside is the real protocol (an unbounded number of keys involved in an unbounded number of sessions) and the right handside is an ideal model (an unbounded number of keys used only once). We cannot handle this equivalence because our replications are visible, but we can encode another property, roughly :

$$\forall k,\ (\mathcal{P} \cup \mathrm{phase}\,1.\ (\ !P(alice) \mid !R(alice)\ ),\ \phi_k,\ 0) \approx (\mathcal{P} \cup \mathrm{phase}\,1.\ (\ !P(bob) \mid !R(bob)\ ),\ \phi_k,\ 0)$$

where $\mathcal{P}$ is a set of processes in phase 0 that represent an unbounded number of passport being read.

We consider the passport protocol of [2] and its unformal description is as follows :

1. $R \to P : GetChallenge$
2. $P \to R : N_t$
3. $R \to P : (M_0 = \{N_r, N_t, K_r\}_{\mathrm{ke}}, \mathrm{mac}(M_0, \mathrm{km}))$
4. P checks the mac. If it does not correspond to the term M_0, $MacError$ is sent.
5. If it corresponds, M_0 is decrypted with ke and the nonce N_t is checked.
6. If N_t is not the nonce created at step 2, $NonceError$ is sent.
7. If it is the same nonce :
8. $P \to R : M'\{N_t, N_r, K_t\}_{\mathrm{ke}}$

ke and km are keys, N_t is a nonce created by the attacker, N_r is a nonce created by the reader, and K_t is an established key created by the passport.

More formally, consider the following signature :

$$\begin{aligned}\Sigma = &\{\mathrm{ag}\,/1; \mathrm{ke}\,/1; \mathrm{km}\,/1; \mathrm{mac}\,/2; \mathrm{enc}\,/2; \mathrm{dec}\,/2; \mathrm{eq}\,/2;\\ &<,>\,/2; \mathrm{proj}_1\,/1; \mathrm{proj}_2\,/1; MacError/0; NonceError/0\} \cup (\Sigma_0^{pub})^-\end{aligned}$$

where the private symbols are $\mathrm{ag}, \mathrm{ke}, \mathrm{km}$, the errors are $MacError$ and $NonceError$ and the destructors are $\mathrm{dec}, \mathrm{eq}$ and proj_i $(i = 1, 2)$. The associated rules are :

$$\mathcal{R} = \{\mathrm{dec}(\mathrm{enc}(x, y), y) \to x\,;\, \mathrm{proj}_i(< x_1, x_2 >) \to x_i\,(i = 1, 2)\,;\, \mathrm{eq}(x, x) \to \mathrm{ok}\,/0\}$$

where $\mathrm{ok} \in (\Sigma_0^{pub})^-$. We also assume that there is some public non-error constant $GetChallenge$.

Now, define the following processes :

The reader part :

$$\begin{array}{l} Reader(x_0) = \\ \mathrm{out}(c'_R, GetChallenge). \\ \mathrm{in}(c'_R, y_{nt}). \\ \mathrm{new}\, N_r.\, \mathrm{new}\, K_r; \\ \mathrm{let}\, y_m = \mathrm{enc}(<< N_r, y_{nt} >, K_r >, \mathrm{ke}(x_0))\, \mathrm{in} \\ \mathrm{out}(c'_R, < y_m, \mathrm{mac}(y_m, \mathrm{km}(x_0)) >). \\ \mathrm{in}(c'_R, y_f) \end{array}$$

The passport part :

$$\begin{array}{l} Passport(x_0) = \\ \mathrm{in}(c'_P, GetChallenge). \\ \mathrm{new}\, N_t.\, \mathrm{out}(c'_P, N_t). \\ \mathrm{in}(c'_P, (x_{me}, x_{mm})). \\ \mathrm{let}\, y_{TestMac} = \mathrm{eq}(x_{mm}, \mathrm{mac}(x_{me}, \mathrm{km}(x_0))\, \mathrm{in} \\ (\\ \mathrm{let}\, y_{TestNonce} = \mathrm{eq}(N_t, \mathrm{proj}_2(\mathrm{dec}(x_{me}, \mathrm{ke}(x_0))))\, \mathrm{in} \\ (\mathrm{let}\, x_{dec} = \mathrm{dec}(x_{me}, \mathrm{ke}(x_0))\, \mathrm{in} \\ \mathrm{let}\, y_{nr} = \mathrm{proj}_1(\mathrm{proj}_1(x_{dec}))\, \mathrm{in} \\ \mathrm{let}\, y_{kr} = \mathrm{proj}_2(x_{dec})\, \mathrm{in} \\ \mathrm{new}\, K_t.\, \mathrm{let}\, z = \mathrm{enc}(<< N_t, y_{nr} >, K_t >, \mathrm{ke}(x))\, \mathrm{in} \\ \mathrm{out}(c'_P, < z, \mathrm{mac}(z, \mathrm{ke}(x_0)) >) \quad) \\ \mathrm{else}\, \mathrm{out}(c'_P, NonceError) \\ ) \\ \mathrm{else}\, \mathrm{out}(c'_P, MacError) \end{array}$$

Then define the restricted protocols :

$$\begin{aligned} P = &!^{c_R^0}_{c'_R} \mathrm{in}(c'_R, \mathrm{ag}(x_0)) Reader(x_0) \\ &|!^{c_P^0}_{c'_P} \mathrm{in}(c'_P, \mathrm{ag}(x_1)) Passport(x_1) \\ &| \,\mathrm{phase}\, 1. \,\mathrm{in}(c_H, < \mathrm{hon}(x_P), \mathrm{hon}(x_Q) >).(!^{c_R^1}_{c'_R} Reader(x_P) | !^{c_P^1}_{c'_P} Passport(x_P)) \end{aligned}$$

$$\begin{aligned} Q = &!^{c_R^0}_{c'_R} \mathrm{in}(c'_R, \mathrm{ag}(x_0)) Reader(x_0) \\ &|!^{c_P^0}_{c'_P} \mathrm{in}(c'_P, \mathrm{ag}(x_1)) Passport(x_1) \\ &| \,\mathrm{phase}\, 1. \,\mathrm{in}(c_H, < \mathrm{hon}(x_P), \mathrm{hon}(x_Q) >).(!^{c_R^1}_{c'_R} Reader(x_Q) | !^{c_P^1}_{c'_P} Passport(x_Q)) \end{aligned}$$

The property we want to check is $\forall k, (P, \phi_k, 0) \approx (Q, \phi_k, 0)$. This property is expressed with our hypothesis, and our main theorem 1 applies : it is sufficient to show $(P, \phi_3, 0) \approx (Q, \phi_3, 0$. In the version here, there is a replay attack : if the attacker chooses $x_P = alice$ and $x_Q = bob$ in phase 1 and replays a message $(me, mac(me, km(alice)))$ that he has seen in phase 0, then he will get $NonceError$ in P but $MacError$ in Q. This can be fixed by using only one constant $Error$ instead of $ErrorMac$ and $ErrorNonce$.

7 Conclusion

My theorem, whose proof is inspired by [6], is more general because it is true for a more general class of protocols, and more precise because it includes completeness. But it holds for names of agents only. So it seems natural to unify these results through a more general theorem. As the soundness result does not rely on specificities of agent names, it is also natural to try to extend it for all kind of atomic data, including public constants or keys. It would also be interesting to understand when the completeness statement is true. Moreover, it would probably be possible to extend the proof to other kinds of else branches that occur inside real protocols. It is also an open problem to bound the number of sessions, but it is a lot more difficult and cannot be handled the same way.

Références

[1] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '01, pages 104–115, New York, NY, USA, 2001. ACM.

[2] Myrto Arapinis, Tom Chothia, Eike Ritter, and Mark Ryan. Analysing unlinkability and anonymity using the applied pi calculus. In *2nd IEEE Computer Security Foundations Symposium (CSF'10)*. IEEE Computer Society Press, 2010.

[3] David Baelde, Stéphanie Delaune, and Lucca Hirschi. Partial order reduction for security protocols. In Luca Aceto and David de Frutos-Escrig, editors, *Proceedings of the 26th International Conference on Concurrency Theory (CONCUR'15)*, Leibniz International Proceedings in Informatics, Madrid, Spain, September 2015. Leibniz-Zentrum für Informatik. To appear.

[4] Bruno Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14)*, pages 82–96, Cape Breton, Nova Scotia, Canada, June 2001. IEEE Computer Society.

[5] Rohit Chadha, Ştefan Ciobâcă, and Steve Kremer. Automated verification of equivalence properties of cryptographic protocols. In Helmut Seidl, editor, *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, Tallinn, Estonia, March 2012. Springer.

[6] Rémy Chrétien, Véronique Cortier, and Stéphanie Delaune. Checking trace equivalence : How to get rid of nonces ? In Peter Ryan and Edgar Weippl, editors, *Proceedings of the 20th European Symposium on Research in Computer Security (ESORICS'15)*, Lecture Notes in Computer Science, Vienna, Austria, September 2015. Springer. To appear.

[7] Rémy Chrétien, Véronique Cortier, and Stéphanie Delaune. Decidability of trace equivalence for protocols with nonces. In *Proceedings of the 28th IEEE Computer Security Foundations Symposium (CSF'15)*, pages 170–184, Verona, Italy, July 2015. IEEE Computer Society Press.

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

[9] Hubert Comon-Lundh and Véronique Cortier. Security properties : Two agents are sufficient. *Science of Computer Programming*, 50(1-3) :51–71, March 2004.

[10] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*. Elsevier, 1990.

[11] Shamir Even and Oded Goldreich. On the security of multi-party ping-pong protocols. In *IEEE Symp. on Foundations of Computer Science*, 2001.

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

A Proofs

A.1 Action-determinism

A.1.1 First property

Our goal is to prove property 1. It gives a way to handle errors in our class of protocols.

Lemma 4 : *Let e be an error. Let C_0 be a configuration and s be a sequence of actions. Let a be a non-error public constant that does not appear in the set of rewriting rules $\mathcal{R}$, in C_0 or in s (it exists as $(\Sigma_0^{pub})^-$ is infinite). Let $E : C_0 \xrightarrow{s} C_1$ be an execution where the rule $[Fail]$ is never used. Then $E' : C_0 \xrightarrow{s'} C_1'$, the execution E where all occuring e outside else branches have been replaced by a, is still an execution. Note that s' is the sequence of actions s where all occurences of e have been replaced by a, and C_1' is C_1 where all occurences of e outside else branches have been replaced by a, including in the frame.*

Proof : Let do the proof by induction on the execution E. If E is the empty execution, then it is obvious because the error e does not appear outside else branches. So assume that E is not the empty execution. Then assume $E : C_0 \xrightarrow{s_0} C_1 \xrightarrow{\alpha} C_2$ such that $C_0 \xrightarrow{s_0'} C_1'$ is an execution. Let α' be α where e has been replaced by a if it occurs. As E was an execution, there exists a process P on which the action α was performed in C_1 by E. If P' is P where each occurence of e outside else branches have been replaced by a, then α' is executable on P' because else branches are not reachable without using the $[Fail]$ rule.

Lemma 5 : *Let C_0 be a configuration. Let s be a sequence of actions. Let $E : C_0 \xrightarrow{s} C_1$ be an execution containing the rules $[Fail]$ and $[Const]$ where the second corresponds to the $\mathrm{out}(c, e)$ action reached by the $[Fail]$ rule for some error constant e. Then $\tilde{E} : C_0 \xrightarrow{\tilde{s}} \tilde{C}_1 = C_1 \cup \{t_i : \mathrm{let}\, x = v \,\mathrm{in}\, P \,\mathrm{else}\, \mathrm{out}(c, e)\}$ ($v\!\downarrow$ is not a message) is an execution, where $\tilde{E}$ is the execution E where the $[Fail]$ and $[Const]$ rules have been removed, and $\tilde{s}$ is the sequence of actions s where the τ and the $\mathrm{out}(c, e)$ corresponding to the $[Fail]$ and $[Const]$ rules have been removed.*

Proof : Obvious by induction on the execution E (it is sufficient to execute E only removing the $[Fail]$ and $[Const]$ rules).

Lemma 6 : *Let C_0 be a configuration and s be a sequence of actions. Let $E : C_0 \xrightarrow{s} C_1$ be an execution. Then there is an execution $\hat{E} : C_0 \xrightarrow{\hat{s}} \hat{C}_1$ that is the same as E, except that each occurence of e for $e \in \Sigma_0^{err}$ not coming from an else branch has been replaced by some non-error public constant a_e.*

Proof : We do the proof by induction on the number of $[Fail]$ rules used. If there is no such rule, we only have to rename each error occuring outside else branches into a new non-error public constant by lemma 4. Assume that we have the result for k such rules. Then consider an execution E with $k+1$ such rules. We can remove a $[Fail]$ rule by lemma 5 to get an execution E' where there are only k $[Fail]$ rules, and then use our induction hypothesis on this execution to get another execution E''. Now, at the point where the $[Fail]$ and $[Const]$ have been removed, it is obvious that it is still possible to get $[Fail]$ and $[Const]$ executed because the correspond $\{t : \mathrm{let}\, v \,\mathrm{in}\, P \,\mathrm{else}\, \mathrm{out}(c, e)\}$ is still reachable (where $v\!\downarrow$ is not a message). So we add them and we get the execution $\hat{E}$.

Lemma 7 : *Let C be a configuration. Let a be some non-error public constant that does not occur in C or in the set of rewriting rules $\mathcal{R}$. Let e be some error that does not occur outside else branches in C. Then the execution E', that is E where a has been replaced by e, is still an execution.*

Proof : It can be done as in lemma 4 : replacing a by e does not change the execution because after changing e was only in $\text{out}(c, e)$ in else branches.

Now prove the main property :

Property 1 : *Let C_P and C_Q be two action-deterministic restricted configurations. If $C_P \not\approx C_Q$, then there exists a trace of non-equivalence* tr *such that there is no error in input or output in* tr*, except perhaps in one* $\text{out}(c, e)$ *at the end (that is, if there is an error e occuring in* tr*, then* $\text{tr} = \text{tr}' \,.\, \text{out}(c, e)$ *for some channel c where there is no error in* tr'*).*

Proof : Assume that $C_P \not\approx C_Q$. Wlog, we can assume that $C_P \not\sqsubseteq C_Q$. We define the size of a trace as the sum of its length and the number of occuring errors (in inputs or in outputs $\text{out}(c, e)$), and we consider a trace of non-inclusion tr of minimal size.

As tr is a trace of non-inclusion, there is an execution $E_P : C_P \stackrel{\text{tr}}{\Longrightarrow} C_P^1$ of minimal length such that for $\forall E_Q : C_Q \stackrel{\text{tr}}{\Longrightarrow} C_Q^1$, we have $C_P^1 \not\sim C_Q^1$.

Assume that there is some error e occuring in tr.

By lemma 6, $\hat{E}_P$ is an execution of C_P with trace $\hat{\text{tr}}$. As $\hat{\text{tr}}$ has no error e in input (error e has been replaced by non-error public constant a_e), error e only occurs in else branches during execution $\hat{E}_P$.

If it exists, consider an execution $C_Q \stackrel{\hat{\text{tr}}}{\Longrightarrow} C_Q'$ (if it does not exist, then the result is true). Then by lemma 7, there is an execution $C_Q \stackrel{\text{tr}}{\Longrightarrow} C_Q^1$ (replacing back a_e by e). As tr is a trace of non-inclusion, we have $C_Q^1 = (\mathcal{Q}, \psi, t) \not\sim C_P^1 = (\mathcal{P}, \phi, t)$. Assume wlog that there is a recipe R such that $R\psi\downarrow$ is a message, but $R\phi\downarrow$ is not, then call R' the recipe R where e has been replaced by a, call ψ' the frame of C_Q and ϕ' the frame of C_P. As a and e does not occur in rewriting rules, we have that $R'\psi'\downarrow$ reduces as a message, but $R'\phi'\downarrow$ doesn't reduce.

So $\hat{\text{tr}}$ is still a trace of non-equivalence between C_P and C_Q. As $\hat{\text{tr}}$ has no error e in input, but only in $\text{out}(c, e)$, and by minimality, it is also the case of tr (else $\hat{\text{tr}}$ has a smaller size than tr).

As tr has only error e in output, and as e only occurs in else branches of C_P and C_Q, at any point of an execution of tr, the error e only occurs in else branches. So by our *ad absurdum* hypothesis, there is an $\text{out}(c, e)$ in $\text{tr} : \text{tr} = \text{tr}_0 \,.\, \text{out}(c, e).\, \text{tr}_1$ with tr_1 not empty.

So denote $\text{tr} = \text{tr}_0 \,.\alpha$. By minimality, tr_0 is executable in C_Q (else it is a smaller trace of non-equivalence). So assume that $\text{tr}_0 = \text{tr}_1 \,.\, \text{out}(c, e).\, \text{tr}_2$ where tr_2 may be empty. As e only occurs in else branches and tr_0 is executable in both C_P and C_Q, it is only executable after a $[Fail]$ operation in both C_P and C_Q. So lemma 5 applies, and $\text{tr}_1 \,.\, \text{tr}_2$ is still executable in C_P (resp. C_Q), and $\text{out}(c, e)$ is executable in the resulting configuration C_P^2 (resp. C_Q^2) whereas α is executable in C_P^2(resp. in C_Q^2 iff α was executable in C_Q). By action-determinism of C_P, $\alpha \neq \text{out}(c, e)$ and $\text{tr}_1 \,.\, \text{tr}_2 \,.\alpha$ is still a trace of non-inclusion (as the resulting frame is the same as this of tr). But $\text{tr}_1 \,.\, \text{tr}_2 \,.\alpha$ is of smaller size than tr which is impossible.

So $\text{out}(c, e)$ does not appear in tr_0. If it appears in tr, then $\text{tr} = \text{tr}_0 \,.\, \text{out}(c, e)$.

A.1.2 Second property

Our goal is to prove property 2. Roughly, it says that for action-deterministic protocols, the trace determines the frame (up to α-renaming).

To do that, we have to introduce a new notion. We say that the execution $C \xrightarrow{s} C'$ is asap if the invisible actions are performed as soon as possible. More formally :

Definition 7 : *Let C, C' be two configurations, s be a sequence of actions. We say that the execution $C \xrightarrow{s} C'$ is asap when :*

— *If $C' \xrightarrow{s'} C''$ and s' is a sequence of silent actions, then $s' = \epsilon$.*

— *If $C \xrightarrow{s_0} C_0 \xrightarrow{\alpha} C_0'$ is a prefix of $C \xrightarrow{s} C'$ with α visible, and if $C_0 \xrightarrow{s_0'} C_0''$ with s_0' silent, then $s_0' = \epsilon$.*

The following lemma says that in our setting, the executions can be asap because once silent actions are available, it is useless to wait to do them.

Lemma 8 : *For any execution $C \xrightarrow{s} (\mathcal{P}, \phi, t)$, there exists an asap execution $C \xrightarrow{s'} (\mathcal{P}', \phi, t)$ with the same visible trace.*

Proof : To prove this lemma, we will prove by induction on the visible trace tr of the execution that for any execution $C \xrightarrow{s} (\mathcal{P}, \phi, t)$, there exists an asap execution $C \xrightarrow{s'} C' = (\mathcal{P}', \phi, t)$ with same visible trace tr such that no visible action is removed if we replace s by s'. More formally, we have toprove that the execution $C \xrightarrow{s'} C'$ has the following properties :

1. If $C' \xrightarrow{s''} C'' = (\mathcal{P}'', \phi', t)$ and s'' is a sequence of silent actions, then $s'' = \epsilon$.
2. If $C \xrightarrow{s_0'} C_0 \xrightarrow{\beta} C_1$ is a prefix of $C \xrightarrow{s'} C'$ with β visible, and if $C_0 \xrightarrow{s''} C_1$ with s'' silent, then $s'' = \epsilon$.
3. If $t' : Q \in \mathcal{P}$ with $t' > t$, or with $t' = t$ and $Q = \beta.Q'$ with β visible, then $t' : Q \in \mathcal{P}'$.

If $\mathrm{tr} = \epsilon$, then the second point of the definition is obvious, because there is no visible action at all. So we only have to prove the first and the last point.

Let prove the first.

Let $C = (\mathcal{P}_0, \phi_0, t_0)$. Let $C \xrightarrow{s} (\mathcal{P}, \phi, t)$ be a silent execution. As there are no visible actions in s, we have that $\phi = \phi_0$ and $t = t_0$. We only have to prove that there is a silent execution $C \xrightarrow{s'} (\mathcal{P}', \phi_0, t_0)$ and no silent action can be executed from this resulting configuration.

Let $t_x : Q \in \mathcal{P}_0$ be a phased process. We can assign an integer $N(t_x : Q)$ to each of them as follows :

— $N(t_x : Q) = 1$ when $t_x < t_0$

— $N(t_x : Q) = 0$ when $t_x > t_0$

— $N(t_0 : 0) = 1$

— $N(t_0 : (Q_1 | Q_2)) = 1 + N(t_0 : Q_1) + N(t_0 : Q_2)$

— $N(t_0 : \mathrm{new}\, n.Q') = 1 + N(t_0 : Q')$

— $N(t_0 : \mathrm{let}\, y = v\, \mathrm{in}\, Q'\, \mathrm{else}\, E) = 1 + max(N(t_0 : Q'), N(t_0 : E))$

— $N(t_0 : \mathrm{phase}\, t_x.Q') = 1 + N(t_x : Q')$

— $N(t_0 : !^c_{c'} P) = 0$.

— $N(t_0 : \alpha.Q') = 0$ when α is visible.

It is obvious that this number is defined and non negative. By extension, we can call $N(\mathcal{P}) = \sum_{t_x : Q \in \mathcal{P}} N(t_x : Q)$. As s and $\mathcal{P}_0$ are finite, $N(\mathcal{P})$ is defined, finite and non negative. Moreover, when $E : (\mathcal{P}_0, \phi_0, t_0) \xrightarrow{s'} (\mathcal{P}_E, \phi_0, t_0)$ is a non-empty silent execution, we have $N(\mathcal{P}_0) > N(\mathcal{P}_E)$. So there is a non-negative minimal such $N(\mathcal{P}_E)$ (that is, an integer N such that $\forall E, N(\mathcal{P}_E) \geq N$ and there is an execution E such that $N(\mathcal{P}_E) = N$).

Assume *ad absurdum* that $N > 0$. Then there is a silent execution $E : C \xrightarrow{s'} (\mathcal{P}_E, \phi_E, t_E)$ such that $N(\mathcal{P}_E) = N > 0$. So there is a $t_x : Q \in \mathcal{P}_E$ such that $N(t_x : Q) > 0$. We are in one of the following cases :

— $N(t_x : Q) = 1$ and $t_x < t_0$
— $N(t_0 : 0) = 1$
— $N(t_0 : (Q_1|Q_2)) = 1 + N(t_0 : Q_1) + N(t_0 : Q_2)$
— $N(t_0 : \text{new}\, n.Q') = 1 + N(t_0 : Q')$
— $N(t_0 : \text{let}\, y = v \,\text{in}\, Q' \,\text{else}\, E) = 1 + max(N(t_0 : Q'), N(t_0 : E))$
— $N(t_0 : \text{phase}\, t_x.Q') = 1 + N(t_x : Q')$

But for each of these cases, there exists a reduction τ such that $t_x : Q \xrightarrow{\tau} t'_x : Q'$ with $N(t_x : Q) \geq 1 + N(t'_x : Q')$ and so there is a E' such that E is a prefix of E' and $N(\mathcal{P}_{E'}) < N$ which is a contradiction. So $N = 0$.

That is, there exists an execution $C \xrightarrow{s'} (\mathcal{P}', \phi_0, t_0)$ and $N(\mathcal{P}') = 0$. Let $t : Q \in \mathcal{P}'$. Then $N(t : Q) = 0$ and $t : Q$ is of one of the following forms :

— $N(t_x : Q) = 0$ when $t_x > t_0$
— $N(t_0 : \alpha.Q') = 0$ when α is visible.

So no silent action can be performed, which proves the first point. But we only have executed silent actions, and we have executed all of them, so visible actions available after s are also available after s', which proves the last point.

Now assume that $\text{tr} = \text{tr}_0.\alpha$ where α is some visible action. The execution $E_0 : C \xRightarrow{\text{tr}} (\mathcal{P}, \phi, t)$ rewrites :

$$C \xrightarrow{s_0} (\mathcal{P}_0, \phi_0, t_0) \xrightarrow{\alpha.s_1} (\mathcal{P}, \phi, t)$$

By induction hypothesis, there exists an asap execution $E : C \xrightarrow{s'_0} (\mathcal{P}'_0, \phi_0, t_0)$ with the same visible trace tr_0 as $C \xrightarrow{s_0} (\mathcal{P}_0, \phi_0, t_0)$. By the third item of the induction hypothesis, α is executable after E on the same $t : Q$ if $\alpha \neq \text{phase}\, t'$ (if $\alpha = \text{phase}\, t'$, the execution only increases the phase). Wlog, if $\alpha = \text{out}(c, w)$ then $t : \text{out}(c, u).Q' \in \mathcal{P}_0 \cap \mathcal{P}'_0$ and :

$$(\mathcal{P}'_0, \phi_0, t_0) \xrightarrow{\alpha} (\mathcal{P}''_0, \phi_0 \cup \{w \triangleright u\} = \phi, t_0 = t)$$

Now, we can apply the case $\text{tr} = \epsilon$ to get an execution $E' : (\mathcal{P}''_0, \phi_1, t_1) \xrightarrow{s'_1} C_{E'} = (\mathcal{P}_{E'}, \phi, t)$ is asap. Consider the execution :

$$E'' : C \xrightarrow{s'_0} (\mathcal{P}'_0, \phi_0, t_0) \xrightarrow{\alpha} (\mathcal{P}''_0, \phi_1, t_1) \xrightarrow{s'_1} C_{E'}$$

Then E_0 and E'' have the same visible trace and the same resulting frame ϕ. Moreover, the first and third item of the induction hypothesis are true because they are true for E'. So we only have to prove the second item. Let β be some visible action of tr. If $\beta \neq \alpha$, then the second item of the induction hypothesis on E gives the result. If $\beta = \alpha$, then the first item of the induction hypothesis on E gives the result.

Lemma 9 : *Let C be an action-deterministic configuration. Let $C \xrightarrow{s_1} (\mathcal{P}_1, \phi_1, t_1)$ and $C \xrightarrow{s_2} (\mathcal{P}_2, \phi_2, t_2)$ be two asap executions with the same visible trace* tr. *Then $(\mathcal{P}, \phi, t) =^{\alpha} (\mathcal{P}', \phi', t')$.*

Proof : First, note the following obvious fact :

Fact 1 : *If* $t : P, t : Q \in \mathcal{Q}$, $(t : P, \emptyset, t) \xrightarrow{\tau} (t : P', \emptyset, t)$ *and* $(t : Q, \emptyset, t) \xrightarrow{\tau} (t : Q', \emptyset, t)$ *then the two following executions result in the same configuration (up to alpha-renaming) :*

$$(\mathcal{Q} = \mathcal{Q}_0 \cup t : P \cup t : Q, \phi, t) \xrightarrow{\tau} (\mathcal{Q}_0 \cup t : P' \cup t : Q, \phi, t) \xrightarrow{\tau} (\mathcal{Q}_0 \cup t : P' \cup t : Q', \phi, t)$$
$$(\mathcal{Q} = \mathcal{Q}_0 \cup t : P \cup t : Q, \phi, t) \xrightarrow{\tau} (\mathcal{Q}_0 \cup t : P \cup t : Q'', \phi, t) \xrightarrow{\tau} (\mathcal{Q}_0 \cup t : P'' \cup t : Q'', \phi, t)$$

Now, let do the proof by induction on tr.

If $\mathrm{tr} = \epsilon$, $\mathcal{P} =^{\alpha} \mathcal{P}'$ because the operations of both executions are exactly the same up to order which implies that resulting configurations are equal up to alpha-renaming from fact 1.

If $\mathrm{tr} = \mathrm{tr}_0 .\alpha$, then assume it is true for tr_0 and consider the following rewriting of executions $E_1 : C \xrightarrow{s_1} (\mathcal{P}_1, \phi_1, t_1)$ and $E_2 : C \xrightarrow{s_2} (\mathcal{P}_2, \phi_2, t_2)$ respectively :

$$E_1 : C \xrightarrow{s_1'} C_1' = (\mathcal{P}_1', \phi_1', t') \xrightarrow{\alpha} C_1'' \xrightarrow{s_1''} C_1$$

$$E_2 : C \xrightarrow{s_2'} C_2' = (\mathcal{P}_2', \phi_2', t') \xrightarrow{\alpha} C_2'' \xrightarrow{s_2''} C_2$$

By induction hypothesis the configurations C_1' and C_2' are equal (up to alpha-renaming). If $\alpha = \mathrm{phase}\, t'$, then after α only the phase has changed. Else, as C is action-deterministic, there exists only one $\beta_1.Q_1 \in \mathcal{P}_1'$ and $\beta_2.Q_2 \in \mathcal{P}_2'$ that correspond to α (channel names are chosen by the attacker in the frame so they cannot be alpha-renamed). So, after $s_1'.\alpha$ and $s_2'.\alpha$ the configurations are still the same both sides (up to alpha-renaming) : the multiset are the same because we have done exactly the same transformation, and the frames are the same because the new frame depends only on the β_1 (which are the same up to alpha-renaming) and of the previous frame that are both equal (upto alpha-renaming). Now, we only do all the τ-actions that can remove some silent operation, so the frames stays equal and the multisets of processes too (up to α-renaming). This proves the lemma.

This allows us to prove property 2 :

Property 2 : *Let* P *be an action-deterministic protocol. Let* tr *be a trace, let* $(\mathcal{P}_1, \phi_1, t_1)$ *and* $(\mathcal{P}_2, \phi_2, t_2)$ *such that* $P \Longrightarrow^{\mathrm{tr}} (\mathcal{P}_1, \phi_1, t)$ *and* $P \Longrightarrow^{\mathrm{tr}} (\mathcal{P}_2, \phi_2, t')$.

Then $\phi_1 = \phi_2$ *(up to alpha-renaming).*

Proof : By lemma 8, there exist two asap executions E_1 and E_2 that correspond respectively to the execution that reaches $(\mathcal{P}_1, \phi_1, t_1)$ and $(\mathcal{P}_2, \phi_2, t_2)$ with tr. By lemma 9 applied to E_1 and E_2, $\phi_1 = \phi_2$ (up to alpha renaming).

A.2 Unbounded number of agents

A.2.1 First steps

We will have to consider frames that are equal up to repetition. The next lemma will help to handle them as if they were equal.

Definition 8 : *Let* $\phi_0 \subset \phi_1$ *be two frames. We say that* ϕ_1 *is an* extension *of* ϕ_0 *if the variables that do not occur in* ϕ_0 *refer to terms that are refered by another variable in* ϕ_0*, that is* $\forall w \in dom(\phi_1) \backslash dom(\phi_0), \exists w' \in dom(phi_0), w\phi_1\downarrow = w'\phi_0\downarrow$. *Let* tr *be a trace where the occuring variables are those of* $dom(\phi_1) \cap bv(\mathrm{tr})$ *and* s *be the corresponding sequence of actions. We say that* $\mathrm{tr}^\star$ *(resp.*

$s^\star$) is a restricted version of tr *(resp. s) if it is the trace* tr *(resp. sequence of action s) where each $w \in dom(\phi_1)\backslash dom(\phi_0)$ has been replaced by some $w' \in dom(\phi_0)$ such that $w'\phi_0\downarrow = w\phi_1\downarrow$. Similarly, for any recipe R, we say that $R^\star$ is a restricted version of R if it is R where each $w \in dom(\phi_1)\backslash dom(\phi_0)$ has been replaced by some $w' \in dom(\phi_0)$ such that $w'\phi_0\downarrow = w\phi_1\downarrow$. When ϕ_1 is an extension of ϕ_0, it is obvious that such restricted versions exist.*

We have the following lemma :

Lemma 10 : *The following properties are true :*

1. *Let $\phi_0 \subset \phi_1$ be two frames (ϕ_1 is not necessarily an extension of ϕ_0). Let* tr *be a trace such that $bv(\text{tr}) \cap dom(\phi_1) = \emptyset$ and such that the variables occuring in* tr *are only those of $dom(\phi_0) \cup bv(\text{tr})$. Then :*
$$(\mathcal{P}, \phi_1, t) \stackrel{\text{tr}}{\Longrightarrow} (\mathcal{P}', \phi_1 \cup \phi, t') \text{ iff } (\mathcal{P}, \phi_0, t) \stackrel{\text{tr}}{\Longrightarrow} (\mathcal{P}', \phi_0 \cup \phi, t')$$
2. *Assume that ϕ_1 is an extension of ϕ_0. Let* tr *be a trace and* $\text{tr}^\star$ *be any restricted version of* tr*. Then :*
$$(\mathcal{P}, \phi_1, t) \stackrel{\text{tr}}{\Longrightarrow} (\mathcal{P}', \phi_1 \cup \phi, t') \text{ iff } (\mathcal{P}, \phi_0, t) \stackrel{\text{tr}^\star}{\Longrightarrow} (\mathcal{P}', \phi_0 \cup \phi, t')$$
3. *Let $\phi_0 \subset \phi_1$ be two frames (ϕ_1 is not necessarily an extension of ϕ_0). Let ϕ and ψ be two frames such that $dom(\phi) = dom(\psi)$ and $dom(\phi) \cap dom(\phi_1) = \emptyset$. Then :*
$$\phi_1 \cup \phi \sim \phi_1 \cup \psi \Rightarrow \phi_0 \cup \phi \sim \phi_0 \cup \psi$$
4. *Assume that ϕ_1 is an extension of ϕ_0. Let ϕ and ψ be two frames such that $dom(\phi) = dom(\psi)$ and $dom(\phi) \cap dom(\phi_1) = \emptyset$. Then :*
$$\phi_0 \cup \phi \sim \phi_0 \cup \psi \text{ iff } \phi_1 \cup \phi \sim \phi_1 \cup \psi$$

Proof : The first and the second item are almost the same, so I only prove the second one, which is the hardest.

So let prove the $\Rightarrow$ part of the second item. We will do an induction on the sequence of actions s (both visible and silent) such that $(\mathcal{P}, \phi_1, t) \xrightarrow{s} (\mathcal{P}', \phi_1 \cup \phi, t')$ and prove that $(\mathcal{P}, \phi_0, t) \xrightarrow{s^\star} (\mathcal{P}', \phi_0 \cup \phi, t')$ where $s^\star$ is a restricted version of s corresponding to $\text{tr}^\star$.

If $s = \epsilon$ then $\mathcal{P}' = \mathcal{P}, \phi = \emptyset$ and $t = t'$ so it is obvious.

Now assume that $s = s'.a$. We have that :
$$(\mathcal{P}, \phi_1, t) \xrightarrow{s'} (\mathcal{P}', \phi_1 \cup \phi', t') \xrightarrow{a} (\mathcal{P}'', \phi_1 \cup \phi'', t'')$$
By induction hypothesis, we have also that :
$$(\mathcal{P}, \phi_0, t) \xrightarrow{(s')^\star} (\mathcal{P}', \phi_0 \cup \phi', t')$$

Assume that a is a silent action. Then assume wlog that it corresponds to a passing let : $\mathcal{P}' = \mathcal{P}'_0 \cup t' : \text{let } x = v \text{ in } P \text{ else } Q$ and $(\mathcal{P}'', \phi_1 \cup \phi'', t'') = (\mathcal{P}'_0 \cup t' : P\{^{v\downarrow}/_x\}, \phi_1 \cup \phi', t')$. Then we can apply exactly the same rule on $(\mathcal{P}', \phi_0 \cup \phi', t')$ and get $(\mathcal{P}'_0 \cup t' : P\{^{v\downarrow}/_x\}, \phi_0 \cup \phi', t')$.

Assume now that a is a visible action. If $a = \text{in}(c, R)$ for some recipe R and some channel c, then $\mathcal{P}' = \mathcal{P}'_0 \cup t' : \text{in}(c, u).P$ with u and $R(\phi_1 \cup \phi')\downarrow$ unifiable and $(\mathcal{P}'', \phi_1 \cup \phi'', t'') = (\mathcal{P}'_0 \cup t' : P\sigma, \phi_1 \cup \phi', t')$

where σ is the most general unifier of u and $R(\phi_1 \cup \phi')\downarrow$. Then, as a is visible, a occurs on $\mathrm{tr}^\star$: the corresponding instruction is $\mathrm{in}(c, R^\star)$. By hypothesis, $R^\star$ is a restricted version of R, so it is obvious that $R^\star(\phi_0 \cup \phi')\downarrow = R(\phi_1 \cup \phi')\downarrow$.

So $R(\phi_0 \cup \phi')$ and u are still unifiable, and their most general unifier is still σ. So :

$$(\mathcal{P}'_0 \cup t' : \mathrm{in}(c, R^\star).P, \phi_0 \cup \phi', t') \xrightarrow{\mathrm{in}(c,R^\star)} (\mathcal{P}' \cup t' : P\sigma, \phi_0 \cup \phi', t')$$

If $a = \mathrm{out}(c, w)$ for some w, then $\mathcal{P}' = \mathcal{P}'_0 \cup t' : \mathrm{out}(c, u).P$ and $(\mathcal{P}'', \phi_1 \cup \phi'', t'') = (\mathcal{P}'_0 \cup t' : P, \phi_1 \cup \phi' \cup \{w \triangleright u\}, t')$. So :

$$(\mathcal{P}'_0 \cup t' : \mathrm{out}(c, w).P, \phi_0 \cup \phi', t') \xrightarrow{\mathrm{out}(c,w)} (\mathcal{P}' \cup t' : P\sigma, \phi_0 \cup \phi' \cup \{w \triangleright u\}, t')$$

The other cases (when $a = \mathrm{sess}(c, ch_i)$ or $a = \mathrm{phase}\, t''$) are obvious and can be handled the same way.

The converse (that is the $\Leftarrow$ part) can be proved the same way.

So now prove the third item.

Let ϕ and ψ be two frames such that $dom(\phi) = dom(\psi)$ and $dom(\phi) \cap dom(\phi_1) = \emptyset$. It is sufficient to prove that :

$$\phi_1 \cup \phi \sqsubset \phi_1 \cup \psi \Longrightarrow \phi_0 \cup \phi \sqsubset \phi_0 \cup \psi$$

Assume that $\phi_1 \cup \phi \sqsubset \phi_1 \cup \psi$.

Let R be a recipe such that $R(\phi_0 \cup \phi)\downarrow \in \mathcal{M}_\Sigma$. Then, $vars(R) \in dom(\phi_0) \cup dom(\phi)$ and $R(\phi_1 \cup \phi)$ is defined because $dom(\phi) \cap dom(\phi_1) = \emptyset$. So $R(\phi_1 \cup \phi)\downarrow = R(\phi_0 \cup \phi)\downarrow \in \mathcal{M}_\Sigma$. As $\phi_1 \cup \phi \sqsubset \phi_1 \cup \psi$, $R(\phi_1 \cup \psi)\downarrow \in \mathcal{M}_\Sigma$. But $vars(R) \in dom(\phi_0) \cup dom(\psi)$ so $R(\phi_0 \cup \psi)\downarrow = R(\phi_1 \cup \psi)\downarrow \in \mathcal{M}_\Sigma$.

The case $R_1 = R_2$ is exactly the same.

Now prove the fourth item. The $\Leftarrow$ part is a consequence of item 3. So we only prove the $\Rightarrow$ part. We assume that $\phi_0 \cup \phi \sqsubset \phi_0 \cup \psi$.

Let R_1, R_2 be two recipes such that $(R_1 = R_2)(\phi_1 \cup \phi)$, $R_1(\phi_1 \cup \phi)\downarrow \in \mathcal{M}_\Sigma$ and $R_2(\phi_1 \cup \phi)\downarrow \in \mathcal{M}_\Sigma$. Then, take $R_1^\star$ and $R_2^\star$ two restricted versions of respectively R_1 and R_2 (it exists). $R_1^\star(\phi_0 \cup \phi)\downarrow = R_1(\phi_1 \cup \phi)\downarrow$, and $R_2^\star(\phi_0 \cup \phi)\downarrow = R_2(\phi_1 \cup \phi)\downarrow$. So $(R_1^\star = R_2^\star)(\phi_0 \cup \phi)$, $R_1^\star(\phi_0 \cup \phi)\downarrow \in \mathcal{M}_\Sigma$. As $\phi_0 \cup \phi \sqsubset \phi_0 \cup \psi$, $(R_1^\star = R_2^\star)(\phi_0 \cup \psi)$. But $R_1^\star(\phi_0 \cup \psi)\downarrow = R_1(\phi_1 \cup \psi)\downarrow$ and $R_2^\star(\phi_0 \cup \psi)\downarrow = R_2(\phi_1 \cup \psi)\downarrow$ so $(R_1 = R_2)(\phi_1 \cup \psi)$.

The case $R(\phi_1 \cup \phi)\downarrow \in \mathcal{M}_\Sigma$ is exactly the same.

It allows us to prove lemma 1 :

Lemma 1 : *Let P and Q be two restricted protocols. Let $k \leq k'$ be two integers. If $(P, \phi_{k'}, 0) \approx (Q, \phi_{k'}, 0)$ then $(P, \phi_k, 0) \approx (Q, \phi_k, 0)$.*

Proof : It is sufficient to prove that if $(P, \phi_{k'}, 0) \sqsubseteq (Q, \phi_{k'}, 0)$ then $(P, \phi_k, 0) \sqsubseteq (Q, \phi_k, 0)$. Assume that $(P, \phi_{k'}, 0) \approx (Q, \phi_{k'}, 0)$. Let tr be such that $(P, \phi_k, 0) \stackrel{\mathrm{tr}}{\Longrightarrow} (\mathcal{P}, \phi_k \cup \phi, t)$. Up to an α-renaming of variables, we can assume that $bv(\mathrm{tr}) \cap dom(\phi_{k'}) = \emptyset$.

Then tr is a trace where only the elements of $bv(\mathrm{tr}) \cup \phi_k$ occur. So by lemma 10 (item 1) we have that $(P, \phi_{k'}, 0) \stackrel{\mathrm{tr}}{\Longrightarrow} (\mathcal{P}, \phi_{k'} \cup \phi, t)$. By trace inclusion, we have that $(Q, \phi_{k'}, 0) \stackrel{\mathrm{tr}}{\Longrightarrow} (\mathcal{Q}, \phi_{k'} \cup \psi, t)$ for some ψ with $\phi_{k'} \cup \phi \sim \phi_{k'} \cup \psi$.

But by lemma 10 (item 1), we have that $(Q, \phi_k, 0) \stackrel{\mathrm{tr}}{\Longrightarrow} (\mathcal{Q}, \phi_k \cup \psi, t)$. Moreover,we have $dom(\phi) = dom(\psi)$ and $dom(\phi) \cap dom(\phi_{k'}) = \emptyset$, so by lemma 10 (item 3), we have that $\phi_k \cup \phi \sim \phi_k \cup \psi$.

Now, our goal is to prove lemma 2, which we recall :

Lemma 2 : *Let $k \geq k_0$ be integers. Let $\rho \in \mathcal{F}^k_{k_0}$. If $(P, \phi_{k_0}, 0) \approx (Q, \phi_{k_0}, 0)$ then $(P, \phi_k\rho, 0) \approx (Q, \phi_k\rho, 0)$.*

Proof : Let $k \geq k_0$ be integers. Let $\rho \in \mathcal{F}^k_{k_0}$. It is sufficient to prove that if $(P, \phi_{k_0}, 0) \sqsubseteq (Q, \phi_{k_0}, 0)$ then $(P, \phi_k\rho, 0) \sqsubseteq (Q, \phi_k\rho, 0)$.

Up to a bijective α-renaming, the names occuring in $\phi_k\rho$ are names of ϕ_{k_0} (because there are at most k_0 different names in $\phi_k\rho$ as $\rho \in \mathcal{F}^k_{k_0}$). Up to a bijective α-renaming of variables, we can assume that the variables in $dom(\phi_k\rho) \cap dom(\phi_{k_0})$ refer to the same terms, that is $\forall w \in dom(\phi_k\rho) \cap dom(\phi_k), w\phi_k\rho = w\phi k_0$. We can also assume that for each variable $w' \in dom(\phi_k\rho) = dom(\phi_k)$, there is a variable $w \in dom(\phi_k) \cap dom(\phi_{k_0})$ such that $(w = w')\phi_k\rho$ (it is possible upto a bijective α-renaming of variables because each names of $\phi_k\rho$ are names of ϕ_{k_0}). We have that $\phi_k\rho$ is an extension of $\phi_k\rho \cap \phi_{k_0}$.

Let tr be a trace such that $(P, \phi_k\rho, 0) \xRightarrow{\text{tr}} (\mathcal{P}', \phi_k\rho \cup \phi, t)$. Let $\text{tr}^\star$ be a restricted version of tr. We apply lemma 10 (second item) and we get :

$$(P, \phi_k\rho \cap \phi_{k_0}, 0) \xRightarrow{\text{tr}^\star} (\mathcal{P}', (\phi_k\rho \cap \phi_{k_0}) \cup \phi, t)$$

Up to a bijective α-renaming, $\text{tr}^\star$ is such that $bv(\text{tr}) \cap dom(\phi_{k_0}) = \emptyset$. Moreover, the variables occuring in tr are only those of $dom(\phi_k\rho \cap \phi_{k_0}) \cap bv(\text{tr})$. So by the lemma 10 (first item), we have :

$$(P, \phi_{k_0}, 0) \xRightarrow{\text{tr}^\star} (\mathcal{P}', \phi_{k_0} \cup \phi, t)$$

But $(P, \phi_{k_0}, 0) \approx (Q, \phi_{k_0}, 0)$, so there is a multiset $\mathcal{Q}'$ and a frame ψ such that :

$$(Q, \phi_{k_0}, 0) \xRightarrow{\text{tr}^\star} (\mathcal{Q}', \phi_{k_0} \cup \psi, t)$$

With $\phi_{k_0} \cup \phi \sim \phi_{k_0} \cup \psi$.

Now, we can apply the converse of lemma 10 (item 1) :

$$(Q, \phi_{k_0} \cap \phi_k\rho, 0) \xRightarrow{\text{tr}^\star} (\mathcal{Q}', (\phi_{k_0} \cap \phi_k\rho) \cup \psi, t)$$

As $\phi_{k_0} \cap \phi_k\rho \subset \phi_k$, the proof of lemma 10 (item 3) applies and $(\phi_{k_0} \cap \phi_k) \cup \phi \sim (\phi_{k_0} \cap \phi_k) \cup \psi$.

As $\text{tr}^\star$ is still a restricted version of tr, we can now apply the lemma 10 (item 2) :

$$(Q, \phi_k\rho, 0) \xRightarrow{\text{tr}^\star} (\mathcal{Q}', \phi_k\rho \cup \psi, t)$$

We apply the lemma 10 (item 3) and we get that $\phi_k\rho \cup \phi \sim \phi_k\rho \cup \psi$.

A.2.2 Bound

Let $(\Sigma, \mathcal{R})$ be a finite theory. We define the *critical number* $\mathfrak{c}$ of $(\Sigma, \mathcal{R})$ as the maximal size of a subset $A \subset \mathcal{R}$ such that all left hand side of rules of A are unifiable. That is :

$$\mathfrak{c} = max\{\#A \mid \exists \sigma, \exists u, \forall (\ell \longrightarrow r) \in A, \ell\sigma = u\}$$

We have the following lemma :

Lemma 11 : *Let $(\Sigma, \mathcal{R})$ be a convergent theory with critical number $\mathfrak{c}$. Let $\mathcal{M}_\Sigma \in \mathcal{T}(\Sigma_c, \hat{\mathcal{N}})$ be a notion of messages. Then $(\Sigma, \mathcal{R})$ is $\mathfrak{c}-$blockable w.r.t. $\mathcal{M}_\Sigma$.*

Proof : Assume $t \in \mathcal{T}(\Sigma_c, \hat{\mathcal{N}})$ in normal form, $t \notin \mathcal{M}_\Sigma$. Then as $\mathcal{T}(\Sigma_c, \hat{\mathcal{N}}) \backslash \mathcal{M}_\Sigma$ is stable by renaming, we can take $N = \emptyset$. Else, there is some destructor in t. Consider a position p where $t = C[g(t_1, \ldots, t_j)]_p$ with g a destructor and $t_1, \ldots, t_j$ constructor terms. Denote $u = g(t_1, \ldots, t_j)$.

Assume that there are renamings ρ_i and rules $\ell_i \to r_i$ that may reduce $u\rho_i$. That is, there is a substitution σ_i such that $u\rho_i = \ell_i\sigma_i$. As u is a ground term, we can assume that $dom(\sigma_i) \subset vars(\ell_i)$. Let denote n_0 be a name and ρ_0 such that $n\rho_0 = n_0$ for any n. We have $u\rho_0 = u\rho_i\rho_0 = \ell_i\sigma_i\rho_0$. As $dom(\sigma_i) \subset vars(\ell_i)$ and as we can assume that the variables used in rules are distinct, we define σ by $x\sigma = x\sigma_i\rho_0$ if $x \in vars(\sigma_i)$ ($\sigma = (\uplus_i \sigma_i)\rho_0$ if we see substitutions as sets : see subsection 2.3). We have $u\rho_0 = \ell_i\sigma$ for each i. So all the ℓ_i are unifiable together : there are at most $\mathfrak{c}$ rules $\ell_i \to r_i$.

Consider the rule $\ell_i \to r_i$. As t is in normal form, u is in normal form and so the rule $\ell_i \to r_i$ does not apply. As $\ell_i \to r_i$ applies to $u\rho_i$ but not to u, it means that there are two positions $p_1 \neq p_2$ such that $\ell_i|_{p_1} = \ell_i|_{p_2} = x$ and two leaf positions $q_1 > p_1$ and $q_2 > p_2$ such that $u|_{q_1} \neq u|_{q_2}$ but $u\rho_i|_{q_1} = u\rho_i|_{q_2}$. $u|_{q_1}$ and $u|_{q_2}$ are names (else renaming can't create equality). Denote $n_i = u|_{q_1}$. For any n_i-adequat renaming ρ, $u\rho|_{q_1} \neq u\rho|_{q_2}$ and so $u\rho|_{p_1} \neq u\rho|_{p_2}$ so $\ell_i \to r_i$ do not apply to $u\rho$.

Also denote N the set of the names n_i. As there is a name by rule and at most $\mathfrak{c}$ rules, N has size at most $\mathfrak{c}$. For any N-adequat renaming ρ, none of the rules $\ell_i \to r_i$ applies to $u\rho$. As the rules $\ell_i \to r_i$ are the only rules that could apply to $u\rho$, we have that $u\rho{\downarrow} = u\rho$. But rules of $\mathcal{R}$ contain exactly one destructor in top level position, so $t\rho{\downarrow} = C\rho[g(t_1\rho{\downarrow}, \ldots, t_j\rho{\downarrow})]$ is not a constructor term and *a fortiori* not a message for any N-adequat renaming ρ.

As rules have exactly one destructor in top level position, note that the maximal subset $A \subset \mathcal{R}$ such that $\mathfrak{c} = \#A$ is a subset of rules that have the same destructor at top level position (because only these rules may be unifiable). So $\mathfrak{c}$ is smaller than the maximal number of rules that have the same destructor at top level position, which is a proof of :

Property 3 : *Let $(\Sigma, \mathcal{R})$ be a theory with $\mathcal{R}$ finite. Let $\mathcal{M}_\Sigma \in \mathcal{T}(\Sigma_c, \hat{\mathcal{N}})$ be a notion of messages. Let $\mathfrak{b}$ be the maximal number of rules that use the same destructor. Then $(\Sigma, \mathcal{R})$ is $\mathfrak{b}$-blockable w.r.t. $\mathcal{M}_\Sigma$.*

A.2.3 Renaming lemma

The following lemma states that when a trace without errors are executable in some configuration C, then this same trace is executable in the configuration $C\rho$ where $\rho \in \mathcal{F}^k_{k_0}$.

Lemma 12 : *Let $k \geq k_0$. Let $\rho \in \mathcal{F}^k_{k_0}$. Let* tr *be a trace where no error occurs. We assume wlog that in such a case, the failing* let *tests are not evaluated (it is useless to evaluate them).*

If $(\mathcal{P}, \phi_k, t) \stackrel{\text{tr}}{\Longrightarrow} (\mathcal{P}', \phi, t')$ then $(\mathcal{P}, \phi_k\rho, t) \stackrel{\text{tr}}{\Longrightarrow} (\mathcal{P}'\rho, \phi\rho, t')$.

Proof : We will prove that if $(\mathcal{P}, \phi_k, t) \xrightarrow{s} (\mathcal{P}', \phi, t')$ then $(\mathcal{P}, \phi_k\rho, t) \xrightarrow{s} (\mathcal{P}'\rho, \phi\rho, t')$ by induction on the sequence of actions s.

If $s = \epsilon$ it is obvious.

If $s = s'.a$, we have $(\mathcal{P}, \phi_k, t) \xrightarrow{s'} (\mathcal{P}', \phi', t') \xrightarrow{a} (\mathcal{P}'', \phi'', t'')$. By induction hypothesis, we also have $(\mathcal{P}, \phi_k\rho, t) \xrightarrow{s'} (\mathcal{P}'\rho, \phi'\rho, t')$.

Assume that a is a silent action. Wlog, we assume that a corresponds to a passing let. Then $\mathcal{P}' = \mathcal{P}'_0 \cup t' : \text{let } x = v \text{ in } P \text{ else } Q$, $\mathcal{P}'' = \mathcal{P}'_0 \cup t' : P\{^{v\downarrow}/_x\}$ (where $v\downarrow$ is a message), $\phi'' = \phi'$ and $t'' = t'$.

Then $(\mathcal{P}'\rho, \phi'\rho, t') = (\mathcal{P}'_0\rho \cup \text{let } x = v\rho \text{ in } P\rho \text{ else } Q, \phi'\rho, t')$. $v\downarrow \rho = v\rho\downarrow$ as messages are constructor terms stable by renaming, so $P\rho\{^{v\rho\downarrow}/_x\} = (P\{^{v\downarrow}/_x\})\rho$. We can do the τ-action and we get :

$$(\mathcal{P}'_0\rho \cup \text{let } x = v\rho \text{ in } P\rho \text{ else } Q, \phi'\rho, t') \xrightarrow{\tau} (\mathcal{P}'_0\rho \cup (P\{^{v\downarrow}/_x\})\rho, \phi'\rho, t')$$

Now, assume that a is a visible action. If $a = \text{in}(c, R)$ then : $\mathcal{P}' = \mathcal{P}'_0 \cup t' : \text{in}(c, u).P$, $\mathcal{P}'' = \mathcal{P}'_0 \cup t' : P\sigma$ (where σ is the most general unifier of $R\phi'\downarrow$ and u), $\phi'' = \phi'$ and $t'' = t'$.

So $\mathcal{P}'\rho = \mathcal{P}'_0\rho \cup t' : \text{in}(c, u\rho).P\rho$. $R\phi'\sigma\downarrow = u\sigma\downarrow$ so $(R\phi'\sigma)\rho\downarrow = (u\sigma)\rho\downarrow$ and $(R\phi'\rho)(\sigma\rho)\downarrow = (u\rho)(\sigma\rho)\downarrow$. So $R\phi'\rho$ and $u\rho$ are unifiable and :

$$(\mathcal{P}'_0\rho \cup t' : \text{in}(c, u\rho).P\rho, \phi'\rho, t') \xrightarrow{\text{in}(c,R)} \mathcal{P}'_0\rho \cup t' : (P\rho)(\sigma\rho), \phi', t')$$

and $(P\rho)(\sigma\rho) = (P\sigma)\rho$.

The other cases are obvious and can be handled the same way.

A.2.4 Difference of recipes lemma

Lemma 13 : *Let R_1, R_2 be two recipes and ϕ a frame such that $R_1\phi\downarrow$ and $R_2\phi\downarrow$ are messages and $(R_1 \neq R_2)\phi$. Then there exists a name n such that any n-adequat renaming ρ verifies that $(R_1 \neq R_2)\phi\rho$.*

Proof : Note that for any renaming ρ, $R_1\phi\rho\downarrow = R_1\phi\downarrow\rho$ as messages are constructor terms. If $(R_1 \neq R_2)\phi$, then either $R_1\phi\downarrow$ and $R_2\phi\downarrow$ don't share the same constructors (and then any renaming will preserve that) or there is at least a leaf position p such that $R_1\phi\downarrow|_p \neq R_2\phi\downarrow|_p$. If there is a constant in one of those positions, this constant won't be modified by any renaming. Else, there are two names $n_1 = R_1\phi\downarrow|_p$ and $n_2 = R_2\phi\downarrow|_p$. Let ρ be any n_1-adequat renaming. Then $R_1\phi\downarrow\rho = R_1(\phi\rho)\downarrow$ and $R_2\phi\downarrow\rho = R_2(\phi\rho)\downarrow$ and so $R_1(\phi\rho)\downarrow|_p = n_1 \neq n_2 = R_2(\phi\rho)\downarrow|_p$.

A.2.5 Main Lemma

Now, we can prove lemma 3 :

Lemma 3 : *Let $(\Sigma, \mathcal{R})$ be any finite convergent theory and $\mathcal{M}_\Sigma$ be a notion of messages such that $(\Sigma, \mathcal{R})$ is $\mathfrak{b}$ blockable w.r.t. $\mathcal{M}_\Sigma$, and denote $k_0 = 2\mathfrak{b} + 1$. Let $k \geq k_0$ be an integer. Let P and Q be two action-deterministic restricted protocols. Then :*

$$[\forall \rho \in \mathcal{F}^k_{k_0}.(P, \phi_k\rho, 0) \approx (Q, \phi_k\rho, 0)] \Rightarrow (P, \phi_k, 0) \approx (Q, \phi_k, 0)$$

Proof : It is sufficient to show that $(P, \phi_k, 0) \not\sqsubseteq (Q, \phi_k, 0) \Longrightarrow \exists \rho \in \mathcal{F}^k_{k_0}(P, \phi_k\rho, 0) \not\sqsubseteq (Q, \phi_k\rho, 0)$.

So assume that $(P, \phi_k, 0) \not\sqsubseteq (Q, \phi_k, 0)$. Let tr be a trace of non-inclusion of minimal length :

$$(P, \phi_k, 0) \overset{\text{tr}}{\Longrightarrow} (\mathcal{P}, \phi, t)$$

As P and Q are action-deterministic [1], we can assume that we are in one of the following cases :

1. There exists a frame ψ (unique by lemma 2) such that $(Q, \phi_k, 0) \overset{\text{tr}}{\Longrightarrow} (\mathcal{Q}, \psi, t)$, but there are two recipes R_1, R_2 such that $R_1\phi\downarrow \in \mathcal{M}_\Sigma$, $(R_1 = R_2)\phi$ but $(R_1 \neq R_2)\psi$.

1. See [5] and property 2

2. There exists a frame ψ (unique by lemma 2) such that $(Q, \phi_k, 0) \stackrel{\text{tr}}{\Longrightarrow} (\mathcal{Q}, \psi, t)$, but there is a recipe R such that $R\phi\downarrow \in \mathcal{M}_\Sigma$, but $R\psi\downarrow \notin \mathcal{M}_\Sigma$.
3. There exists no frame ψ such that $(Q, \phi_k, 0) \stackrel{\text{tr}}{\Longrightarrow} (\mathcal{Q}, \psi, t)$.

In the first and second cases, the disequivalence comes from the frame so by minimality of tr, the last action is an output. In particular, it is not an error and by property 1, there is no output of error in tr.

First case : $R_1\phi\downarrow \in \mathcal{M}_\Sigma$, $(R_1 = R_2)\phi$ but $(R_1 \neq R_2)\psi$. By lemma 13, there exists a name n such that for any n-adequat renaming ρ we have $(R_1 \neq R_2)\psi\rho$. If $n \in \mathcal{N}$, then any renaming $\rho \in \mathcal{F}^k_{k_0}$ is n-adequat. Else, $n \in \mathcal{A}_k$. Wlog, assume $n = ag^H_1$. Then call ρ the renaming such that $ag^H_i\rho = ag^H_2$ for each $k \geq i \geq 2$ and $ag^D_i\rho = ag^D_1$ for each $i \leq k$. We have $\rho \in \mathcal{F}^k_{k_0}$ as $k_0 = 2\mathfrak{b}+1 \geq 3$. After lemma 12, we have $(P, \phi_k\rho, 0) \stackrel{\text{tr}}{\Longrightarrow} (\mathcal{P}\rho, \phi\rho, t)$ and $(Q, \phi_k\rho, 0) \stackrel{\text{tr}}{\Longrightarrow} (\mathcal{Q}\rho, \psi\rho, t)$. We have still $(R_1 = R_2)\phi\rho$ (renaming can only create more equalities and messages are stable by renaming) and $(R_1 \neq R_2)\psi\rho$ (lemma 13).

Second case : $R\psi\downarrow \notin \mathcal{M}_\Sigma$: there is a set of at most $\mathfrak{b}$ names N such that for any N-adequat renaming ρ we have $R\psi\rho\downarrow \notin \mathcal{M}_\Sigma$. If $n \in \mathcal{N}$, then any renaming $\rho \in \mathcal{F}^k_{k_0}$ is n-adequat. Else, $n \in \mathcal{A}_k$. N contains k_H honest agents and k_D dishonest ones with $k_H + k_D \leq \mathfrak{b}$. Wlog, we can assume that $N = \{ag^H_1, \ldots, ag^H_{k_H}, ag^D_1, \ldots, ag^D_{k_D}\}$. Define ρ such that $ag^H_i\rho = ag^H_{k_H+1}$ for each $k \geq i \geq k_H + 1$ and $ag^D_i\rho = ag^D_{k_D+1}$ for each $k \geq i \geq k_D + 1$. We have $\rho \in \mathcal{F}^k_{k_0}$ as $k_0 = 2\mathfrak{b} + 1 \geq \mathfrak{b} + 1 \geq max(k_H, k_D) + 1$. After lemma 12, we have $(P, \phi_k\rho, 0) \stackrel{\text{tr}}{\Longrightarrow} (\mathcal{P}\rho, \phi\rho, t)$ and $(Q, \phi_k\rho, 0) \stackrel{\text{tr}}{\Longrightarrow} (\mathcal{Q}\rho, \psi\rho, t)$. We have still $R\phi\rho\downarrow \in \mathcal{M}_\Sigma$ (messages are constructor terms stable by renaming) and $R\psi\rho\downarrow$ (lemma 11).

Last case : tr doesn't pass in Q. By minimality, the non-passing action is the last one : $\text{tr} = \text{tr}_0 .\alpha$ and by property 1 tr_0 does not contain any output of error. Assume that there is some $\rho \in \mathcal{F}^k_{k_0}$ such that tr passes in $(Q, \phi_k\rho, 0)$ (if not, we get the result). There are four subcases :

— The action α is not an error and is not executed in $(Q, \phi_k, 0)$ because there is a unique failing let (by action-determinism) that prevents it.
— The action α is an error and is not executed in $(Q, \phi_k, 0)$ because the corresponding let doesn't fail.
— The action α is an error and is not executed in $(Q, \phi_k, 0)$ because there is a unique (by action-determinism) failing let that prevents it.
— The action α is an input ($\alpha = \text{in}(c, R)$) and is not executed in $(Q, \phi_k, 0)$ because $R\psi\downarrow$ does not unify with the unique (by action-determinism) corresponding $\text{in}(c, v)$.

First subcase. Consider the failing let : $\text{let}\, x = v \,\text{in}\, Q' \,\text{else}\, E$. Then by lemma 11, there is a set N of $\mathfrak{b}$ names such for that any N-adequat renaming ρ, we have $v\rho\downarrow \notin \mathcal{M}_\Sigma$. We can define such a renaming $\rho \in \mathcal{F}^k_{k_0}$ as in previous case. Then by lemma 12 we get that tr passes in $(P, \phi_k\rho, 0)$ but not in $(Q, \phi_k\rho, t)$.

Second subcase. As the corresponding let passes in $(Q, \phi_k, 0)$, it will pass in each $(Q, \phi_k\rho, 0)$ as messages are stable by renaming, so this subcase cannot happen.

Third subcase. α is the output of an error and $\text{tr}_0 .\alpha$ is executed in $(P, \phi_k, 0)$ so there is a failing test in P. By lemma 11, this test may be preserved failing with the set N_1 of $\mathfrak{b}$ names (that is, the test fails in $(P, \phi_k, 0)$ for any N_1-adequat renaming ρ). There is a failing let that prevents α to be executed in the Q side : by lemma 11, there is a set N_2 of $\mathfrak{b}$ names such that this test may be preserved failing for any N_2-adequat renaming. Define $N = N_1 \cup N_2$. If $n \in \mathcal{N}$, then any renaming $\rho \in \mathcal{F}^k_{k_0}$ is n-adequat. Else,

$n \in \mathcal{A}_k$. N contains k_H honest agents and k_D dishonest ones with $k_H + k_D \leq 2\mathfrak{b}$. Wlog, we can assume that $N = \{ag_1^H, \ldots, ag_{k_H}^H, ag_1^D, \ldots, ag_{k_D}^D\}$. Define ρ such that $ag_i^H\rho = ag_{k_H+1}^H$ for each $k \geq i \geq k_H + 1$ and $ag_i^D\rho = ag_{k_D+1}^D$ for each $k \geq i \geq k_D + 1$. We have $\rho \in \mathcal{F}_{k_0}^k$ as $k_0 = 2\mathfrak{b} + 1 \geq max(k_H, k_D) + 1$. By lemma 12 we get that tr_0 passes in both $(P, \phi_k\rho, 0)$ and $(Q, \phi_k\rho, 0)$, the final error α is executable in the P side (because the corresponding let is still failing as ρ is N_1-adequat) but not in the Q side (because the preventing let is still failing as ρ is N_2-adequat).

Last subcase. Assume that $R\psi\downarrow$ does not unify with u, but $R\psi\rho\downarrow$ does. Note that as we are not in the second case and $R\phi\downarrow$ is a message, we have that $R\psi\downarrow$ is a message. In particular, it is a constructor term and $R\psi\rho\downarrow = R\psi\downarrow\rho$. It means that there is a variable x at two leaf positions p and q in u such $R\psi\downarrow|_p \neq R\psi\downarrow|_q$. By lemma 13, there is a name n such that any n-adequat renaming ρ verifies $R\psi\rho\downarrow|_p \neq R\psi\rho\downarrow|_q$. If $n \in \mathcal{N}$, then any renaming $\rho \in \mathcal{F}_{k_0}^k$ is n-adequat. Else, $n \in \mathcal{A}_k$. Wlog, assume $n = ag_1^H$. Then call ρ the renaming such that $ag_i^H\rho = ag_2^H$ for each $k \geq i \geq 2$ and $ag_i^D\rho = ag_1^D$ for each $i \leq k$. We have $\rho \in \mathcal{F}_{k_0}^k$ as $k_0 = 2\mathfrak{b} + 1 \geq 3$. By lemma 12, we get that tr_0 still passes in both $(P, \phi_k\rho, 0)$ and $(Q, \phi_k\rho, 0)$, but the final input only passes in the P side (it does not pass in the Q side because the unique corresponding input is not unifiable with $R\psi\downarrow$ as ρ is n-adequat).

B Counter-examples

In this section, I explain some of our hypothesis. First of all, I explain why we cannot consider protocols with general else branches. Another restriction of our model is that our rules use destructor on top-level positions only : in particular, $g(t_1, \ldots, t_n)$ does not reduce if $t_1, \ldots, t_n$ are not constructor terms. There is also some counter example when we remove this hypothesis. Finally, our result is not true for non action-deterministic protocols.

B.1 Else branches

We first explain why our reduction result does not hold in the presence of (non-trivial) else branches. This is a more formal version of the counter-example presented in section 6.1.

Assume that $\{\mathrm{enc}/2, \mathrm{dec}/2, <,> /2, \mathrm{proj}_1/1, \mathrm{proj}_2/1, \mathrm{eq}/2, ok/0, \mathrm{ag}/1\} \subset \Sigma$. Recall that $\mathcal{R}$ is a set of rewritting rules that apply on constructor terms only. Assume that $\mathcal{R}$ contains rules naturally associated to the specified elements of Σ (e.g. $\mathrm{eq}(x, y) \longrightarrow ok$). We assume also that there is some special public constant *end*.

Consider the following protocol :

The three first parts of the protocol are used to approve some lists, represented as a pair of a head element and a tail list. A list can be approved iff it only contains pairwise distinct public keys. P_{init} is used to allow all list containing only one element.

$$
\begin{aligned}
&P_{init} = \\
&\mathrm{in}(c_1, < \mathrm{ag}(x), end >) \\
&\mathrm{out}(c_1, \mathrm{enc}(< \mathrm{ag}(x), end >, k_{approved}))
\end{aligned}
$$

Then, P_{diff} is used to check that the elements of a pair are distinct.

$$
\begin{array}{l}
P_{diff} = \\
\mathrm{in}(c_2, < \mathrm{ag}(x), \mathrm{ag}(y) >). \\
\mathrm{let}\, z = \mathrm{eq}(x, y)\, \mathrm{in}\, 0 \\
\mathrm{else}\, \mathrm{out}(c_2, \mathrm{enc}(< x, y >, k_{diff}))
\end{array}
$$

Now, $P_{decompose}$ is used to get bigger lists. That is, when a list $< x, y >$ and $< x', y >$ are approved, then $< x, < x', y >>$ is also approved if $x \neq x'$. Indeed, a pair of elements of $< x, < x', y >>$ may be a pair of elements of y, or x and an element of y, or x' and an element of y, or the pair $< x, x' >$.

$$
\begin{array}{l}
P_{decompose} = \\
\mathrm{in}(c_3, \mathrm{enc}(< x, x' >, k_{diff})). \\
\mathrm{in}(c_3, < \mathrm{enc}(< x, y >, k_{approved}), \mathrm{enc}(< x', y >, k_{approved}) >) \\
\mathrm{out}(\mathrm{enc}(< x, < x', y >>, k_{approved}))
\end{array}
$$

A more formal version of the PCP part :

$$
\begin{array}{l}
P_{PCP} = \\
\mathrm{out}(c_4, \mathrm{enc}(<< u_{init}, v_{init} >, end >, k_{PCP})) \\
\mathrm{in}(c_4, \mathrm{ag}(x_{new})) \\
\mathrm{in}(c_4, \mathrm{enc}(<< x, y >, \ell >, k_{PCP})) \\
\mathrm{out}(c_4, \mathrm{enc}(<< xu_1, yv_1 >, < \mathrm{ag}(x_{new}), \ell >>, k_{PCP})) \\
\ldots \\
\mathrm{out}(c_4, \mathrm{enc}(<< xu_n, yv_n >, < \mathrm{ag}(x_{new}), \ell >>, k_{PCP}))
\end{array}
$$

For readability purposes, we separate the checking part from the general one. There are two versions of it (one in P and one in Q). We assume that yes and no are public constants.

$$
\begin{array}{l}
P_{check} = \\
\mathrm{in}(c_5, \mathrm{enc}(<< x, x >, x_\ell >, k_{PCP})) \\
\mathrm{in}(c_5, \mathrm{enc}(x_\ell, k_{approved})) \\
\mathrm{out}(c_5, yes)
\end{array}
$$

$$
\begin{aligned}
&Q_{check} = \\
&\mathrm{in}(c_5, \mathrm{enc}(<< x, x >, x_\ell >, k_{PCP})) \\
&\mathrm{in}(c_5, \mathrm{enc}(x_\ell, k_{approved})) \\
&\mathrm{out}(c_5, no)
\end{aligned}
$$

Then define :

$$P = !^{c'_1}_{c_1} P_{init} | !^{c'_2}_{c_2} P_{diff} | !^{c'_3}_{c_3} P_{main} | !^{c'_4}_{c_4} P_{PCP} | !^{c'_5}_{c_5} P_{check}$$

$$Q = !^{c'_1}_{c_1} P_{init} | !^{c'_2}_{c_2} P_{diff} | !^{c'_3}_{c_3} P_{main} | !^{c'_4}_{c_4} P_{PCP} | !^{c'_5}_{c_5} Q_{check}$$

Recall that the frame ϕ_k has been defined in section 3. We have :

1. $\forall k, (P, \phi_k, 0) \approx (Q, \phi_k, 0)$ iff the corresponding PCP instance has no solution.
2. If $(P, \phi_k, 0) \not\approx (Q, \phi_k, 0)$ for some k, then the element matching variable x_ℓ of P_{check} and Q_{check} is a list of $N-1$ public keys where N is the number of tiles of a solution of the corresponding PCP instance.
3. The adversary can built $\mathrm{enc}(x_\ell, k_{approved})$ from x_ℓ iff it contains only distinct public keys.

So we need that $2k \geq N-1$ (because there are $2k$ public keys in ϕ_k), where N is the number of tiles of a solution of the corresponding PCP instance, which is not calculable.

Note that this example is also a counter-example in the case of reachability properties : instead of equivalence, we could have asked for the secret of a constant that would take the place of *yes/no*.

B.2 Pure equational theories

We want to explain why we need to consider only theories with destructors on constructor terms.

Consider $\Sigma = \{\mathrm{enc}/2; \mathrm{dec}/2; m/0\}$ and $\mathcal{R} = \{\mathrm{dec}(\mathrm{enc}(x, y), y) \longrightarrow x\}$. Wlog, we can assume there is another function symbol decenc /3 with a rule $\mathrm{decenc}(x, y, y) \longrightarrow x$: indeed, define $\mathrm{decenc}(x, y, z) = \mathrm{dec}(\mathrm{enc}(x, y), z)$. Here, we assume a standard equational theory, that is that the rules also apply when subterms have destructors. For example, $\mathrm{decenc}(m, \mathrm{dec}(a, b), \mathrm{dec}(a, b))$ reduces into m.

Then, let $n_1, n'_1, \ldots, n_{k+1}, n'_{k+1}$ be some names. We define $t^1_{P:n_1,n'_1,\ldots,n_{k+1},n'_{k+1}}$ and $t^1_{Q:n_1,n'_1,\ldots,n_{k+1},n'_{k+1}}$ inductively as follows :

— $t^1_{P:n_1,n'_1} = \mathrm{decenc}(m, n_1, n'_1)$ and $t^1_{Q:n_1,n'_1} = \mathrm{decenc}(m, n'_1, n_1)$

— Denote $t^k_{P:n_1,n'_1,\ldots,n_k,n'_k} = \mathrm{decenc}(m, t_1, t'_1)$ for some t_1 and t'_1, and $t^k_{Q:n_1,n'_1,\ldots,n_k,n'_k} = \mathrm{decenc}(m, t_2, t'_2)$ for some t_2 and t'_2. We have :

$$t^{k+1}_{P:n_1,n'_1,\ldots,n_{k+1},n'_{k+1}} = \mathrm{decenc}(m, \mathrm{decenc}(n_{k+1}, t_1, t'_1), \mathrm{decenc}(n'_{k+1}, t_1, t'_1))$$

and :

$$t^{k+1}_{Q:n_1,n'_1,\ldots,n_{k+1},n'_{k+1}} = \mathrm{decenc}(m, \mathrm{decenc}(n_{k+1}, t_2, t'_2), \mathrm{decenc}(n'_{k+1}, t_2, t'_2))$$

Then, it is easy to show that for all ρ, $t^k_{P:n_1,n'_1,\ldots,n_k,n'_k}\rho \neq t^k_{Q:n_1,n'_1,\ldots,n_k,n'_k}\rho$ iff $n_1\rho \neq n'_1\rho, \ldots, n_k\rho \neq n'_k\rho$.

So, let K be an integer and N be a set of terms of size K. Define $n_1, n'_1, \ldots, n_k, n'_k$ (with $k = \frac{K(K-1)}{2}$) such that the pairs (n_i, n'_i) are all the non-oriented pairs of distinct elements of N. Then define :

$$t_P^N = t_{P:n_1,n'_1,\ldots,n_k,n'_k}^k$$

$$t_Q^N = t_{P:n_1,n'_1,\ldots,n_k,n'_k}^k$$

Then for any ρ, $t_P^N\rho = t_Q^N\rho$ iff $N\rho$ has cardinal K.

The existence of such terms make lemma 13 false. Moreover, lemma 11 is also false : the term $\text{decenc}(m, t_P^N, t_Q^N)\rho$ doesn't reduce iff $N\rho$ has cardinal K which can be chosen arbitrary big. We can imagine protocols P and Q that builds terms t_P^N and t_Q^N inductively under encryption, where N is a set of names of agents, and such that $(P, \phi_k, t) \not\approx (Q, \phi_k, t)$ iff $t_P^N \neq t_Q^N$, that is iff $k \geq K$. But as this K may be chosen arbitrarily, the bound depends of the protocol.

B.3 Action-determinism

Recall that ϕ_k has been introduced in section 3. We design here protocols P and Q without else branches, such that $(P, \phi_k, 0) \not\approx (Q, \phi_k, 0)$ for some big enough k but such that no bound on k is computable. That is, $\forall k, (P, \phi_k, 0) \approx (Q, \phi_k, 0)$ is not always true, but it is not possible to compute (from the equational theory or from the protocols) a k_0 such that $\forall k, (P, \phi_k, 0) \approx (Q, \phi_k, 0) \Rightarrow (P, \phi_{k_0}, 0) \approx (Q, \phi_{k_0}, t)$. So no extension of theorem 1 can be proved.

Let $(u_i, v_i)_i$ be an instance of PCP. We have public constants yes, no, end, public channels $c_1, \ldots, c_5$, and private keys k_{PCP}, K, K'.

Here, the idea is the same as in subsection B.1 for else branches. There are no else branch, but the construction $(\text{if } x = true \text{ then } P | \text{ if } x = false \text{ then } Q)$ plays a similar role. As in subsection B.1, the three first processes are used to approve some list. So the first step is to approve the smallest lists (here encryption is randomized by a nonce n).

$$P_{init} = \text{in}(c_1, < x, end >).\, \text{new}\, n.\, \text{out}(c_1, \text{senc}(<< yes, n >, < x, end >>, K')))$$

Now, we cannot check that two element of a pair are different, but we ensure that the output *may* be different for equal inputs. Note that the pair $< x, x >$ can pass in the second branch (but trace inclusion ask some property to be true for *any* execution, that is whatever branch the input passes). But only a pair of equal elements may be unapproved (marked by "no").

$$\begin{aligned} &P_{diff} = \\ &(\text{in}(c_2, < x, x >).\, \text{new}\, n.\, \text{out}(c_2, \text{senc}(<< no, n >, < x, x >>, K))) \\ &|(\text{in}(c_2, < x, y >).\, \text{new}\, n.\, \text{out}(c_2, \text{senc}(<< yes, n >, < x, y >>, K))) \end{aligned}$$

The third part is used to approve the complete list. A list is unapproved (marked by "no") iff one of its sublist is unapproved, or if the studied pair is unapproved. Note that here, there is no non-determinism : once known the marks *yes* and *no* of the inputs, there is only one possible result.

The protocol P_{main} can be read as a truth table : there are three four parallel branches that represent a raw, whereas the three inputs represent the variable columns and the final output represent the result column. The first input gives informations on the pair x, y (has it been approved ?), the second input gives informations on the list $< x, z_\ell >$ and the last input gives informations on the list $< y, z_\ell >$. From these informations, the list $< x, < y, z_\ell >>$ is computed and approved or not according to the following rules :

— First raw : if the pair $< x, y >$ (first input) has been unapproved, then the output is not approved.
— Second raw : if the list $< x, z_\ell >$ (second input) has been unapproved, then the output is not approved.
— Third raw : if the list $< y, z_\ell >$ (third input) has been unapproved, then the output is not approved.
— Last raw : if the pair $< x, y >$, the lists $< x, z_\ell >$ and $< y, z_\ell >$ have been approved, then the output is approved.

The P_{main} process is as follows :

$$
\begin{aligned}
&P_{main} = \\
&(\\
&\mathrm{in}(c_3, \mathrm{senc}(<< no, z_n >, < x, y >>, K)).\, \mathrm{in}(c_3, \mathrm{senc}(<< z_1, z'_n >, < x, z_\ell >>, K')). \\
&\mathrm{in}(c_3, \mathrm{senc}(<< z_2, z''_n >, < y, z_\ell >>, K')).\, \mathrm{new}\, n.\, \mathrm{out}(c_3, \mathrm{senc}(<< no, n >, < x, < y, z_\ell >>>, K')) \\
&| \\
&\mathrm{in}(c_3, \mathrm{senc}(<< z_{bool}, z_n >, < x, y >>, K)).\, \mathrm{in}(c_3, \mathrm{senc}(<< no, z'_n >, < x, z_\ell >>, K')). \\
&\mathrm{in}(c_3, \mathrm{senc}(<< z_2, z''_n >, < y, z_\ell >>, K')).\, \mathrm{new}\, n.\, \mathrm{out}(c_3, \mathrm{senc}(<< no, n >, < x, < y, z_\ell >>>, K')) \\
&| \\
&\mathrm{in}(c_3, \mathrm{senc}(<< z_{bool}, z_n >, < x, y >>, K)).\, \mathrm{in}(c_2, \mathrm{senc}(<< z_1, z'_n >, < x, z_\ell >>, K')). \\
&\mathrm{in}(c_3, \mathrm{senc}(<< no, z''_n >, < y, z_\ell >>, K')).\, \mathrm{new}\, n.\, \mathrm{out}(c_3, \mathrm{senc}(<< no, n >, < x, < y, z_\ell >>>, K')) \\
&| \\
&\mathrm{in}(c_3, \mathrm{senc}(<< yes, z_n >, < x, y >>, K)).\, \mathrm{in}(c_2, \mathrm{senc}(<< yes, z'_n >, < x, z_\ell >>, K')). \\
&\mathrm{in}(c_3, \mathrm{senc}(<< yes, z''_n >, < x, z_\ell >>, K')).\, \mathrm{new}\, n.\, \mathrm{out}(c_3, \mathrm{senc}(<< yes, n >, < x, < y, z_\ell >>>, K')) \\
&)
\end{aligned}
$$

The following process is exactly the same as in B.1. It creates a link with a PCP instance.

$$
\begin{aligned}
&P_{PCP} = \\
&\mathrm{out}(c_4, \mathrm{enc}(<< u_{init}, v_{init} >, end >, k_{PCP})) \\
&\mathrm{in}(c_4, \mathrm{ag}(x_{new})) \\
&\mathrm{in}(c_4, \mathrm{enc}(<< x, y >, \ell >, k_{PCP})) \\
&\mathrm{out}(c_4, \mathrm{enc}(<< xu_1, yv_1 >, < \mathrm{ag}(x_{new}), \ell >>, k_{PCP})) \\
&\dots \\
&\mathrm{out}(c_4, \mathrm{enc}(<< xu_n, yv_n >, < \mathrm{ag}(x_{new}), \ell >>, k_{PCP}))
\end{aligned}
$$

We now explain how P and Q differ. The second input in P can only be done if the list is marked with "no" (that is, if the list is "unapproved") whereas the corresponding input can be done in Q whatever the mark. So the protocols will not be equivalent if it is possible to get some list that may not be unapproved in the P side : that will only be possible if all of its elements are distincts.

$$P_{check} = \\ \mathrm{in}(c_4, \mathrm{enc}(<< x, x >, x_\ell >, k_{PCP})) \\ \mathrm{in}(c_4, \mathrm{enc}(<< no, z_n >, x_\ell >, K'))$$

$$Q_{check} = \\ \mathrm{in}(c_4, \mathrm{enc}(<< x, x >, x_\ell >, k_{PCP})) \\ \mathrm{in}(c_4, \mathrm{enc}(<< z_{bool}, z_n >, x_\ell >, K'))$$

Then define :

$$P = !_{c_1}^{c'_1} P_{init} | !_{c_2}^{c'_2} P_{diff} | !_{c_3}^{c'_3} P_{main} | !_{c_4}^{c'_4} P_{PCP} | !_{c_5}^{c'_5} P_{check}$$

$$Q = !_{c_1}^{c'_1} P_{init} | !_{c_2}^{c'_2} P_{diff} | !_{c_3}^{c'_3} P_{main} | !_{c_4}^{c'_4} P_{PCP} | !_{c_5}^{c'_5} Q_{check}$$

Recall that the frame ϕ_k has been defined in section 3, and note the following points :

1. When the PCP instance has a solution, there is an attack. We build the solution with a list of public keys pairwise distincts, then we use Q_{diff} only with pairs of nonces that are different, so the only possible execution in the P side is to do the same. Then we get senc($<< yes, z_n >, x_\ell >, K'$) both sides. It passes the input in the Q_{check} side but not in the P_{check} side.
2. It is obvious that everything that can be done in P_{check} can also be done in Q_{check}. As the only difference between P and Q is between P_{check} and Q_{check}, everything that can be done in the P side can be done in the Q side. That is, $\forall k, (P, \phi_k, 0) \sqsubseteq (Q, \phi_k, 0)$.
3. For ϕ_k with $2k \leq N$ where N is the smallest size of the solution of the PCP problem, when checking the list, we have to use P_{diff} on a pair x, x (because we test all pairs and we cannot have all pairs distincts if the public keys are not pairwise distincts). So, in the P side, we can get the "no" with our pair, and thus at the end, we can make the input, and so we can make no difference between P and Q, because $(P, \phi_k, 0) \sqsubseteq (Q, \phi_k, 0)$ by 2 and $(Q, \phi_k, 0) \sqsubseteq (P, \phi_k, 0)$ for $2k \leq N$.

So we have to use a k such that $2k \geq N$, but N is not computable.